目次

改訂情報

変更年月日	変更内容
2012-10-01	初版
2013-02-01	第2版 下記を追加しました。 ■ 「 *前処理* 」
2013-08-20	第3版 下記を変更しました。 ■ 「 *前処理* 」 ■ 「 *後処理* 」
2014-01-01	第4版 下記を変更・追加しました。 ■ 「 *入力チェックプログラム* 」 ■ 「 *前処理* 」 ■ 「 *後処理* 」 ■ 「 *採番プログラム* 」 ■ 「 *PDF出力処理* 」 ■ 「 *テンプレートの格納先* 」
2015-04-01	第5版 下記を変更・追加しました。 ■ 「 *採番プログラム* 」に採番の体系についての説明を追加しました。
2015-08-01	第6版 下記を追加しました。 ■ 「 *ワークフロー案件処理API* 」を追加しました。 ■ 「 *ワークフロー案件処理 Web API* 」を追加しました。 ■ 「 *IM-Workflow で、申請、承認前に任意のプログラムを実行する方法* 」を追加しました。
2015-12-01	第7版 下記を変更・追加しました。 ■ 「 *APIの仕様を確認する* 」に追加公開のAPIの説明を追加しました。 ■ 「 *登録データ情報管理API* 」のスクリプト開発モデル データの取得メソッドの記載内容を変更しました。 ■ 「 *ジョブスケジューラ・非同期処理機能での実装手順* 」を追加しました。
2016-08-01	第8版 下記を変更・追加しました。 ■ 「 *APIの仕様を確認する* 」を変更しました。 　■ 「 *アプリケーションデータ（ファイルアップロードアイテムデータ）* 」のサンプルコード 　■ 提供APIのうち、一時保存案件の操作に関するAPIの説明を追加しました。 ■ 「 *ジョブスケジューラ・非同期処理機能での実装手順* 」に Office 365 連携 のPDF出力利用に関する注意事項を追加しました。 ■ 「 *入力チェックプログラム* 」の説明、およびサンプルコードを変更しました。 ■ 以下のページ内の表を変更しました。 　■ 「 *入力チェックプログラム* 」 　■ 「 *後処理* 」 　■ 「 *前処理* 」 　■ 「 *採番プログラム* 」 ■ IM-BPM のリリースに伴い、BIS作成種類「BPM」を「BISフロー」に変更しました。

変更年月日	変更内容
2016-12-01	第9版 下記を変更・追加しました。 ■ 「 *後処理* 」にユーザプログラム内でエラー発生時に、画面に指定メッセージを表示する方法の説明を追加しました。 ■ 「 *前処理* 」に、返却値で画面アイテム「「ファイルアップロード」データを指定する方法に関する説明を追加しました。 ■ 「 *ファイルアップロードアイテム API* 」を追加しました。 ■ 「 *入力チェックプログラム* 」に一時保存時の入力チェックプログラムの実行に関する説明を追加しました。
2017-04-01	第10版 下記を変更しました。 ■ 「 *入力チェックプログラム* 」の説明を変更しました。 ■ 「 *後処理* 」に「後処理プログラム-一時保存」機能の説明を追加しました。
2017-08-01	第11版 下記を変更しました。 ■ 以下のページに案件処理API/Web APIを利用してアプリケーションデータを更新する場合の注意事項を追加しました。 　■ 「 *注意事項* （ワークフロー案件処理API）」 　■ 「 *注意事項* （ワークフロー案件処理Web API）」
2017-12-01	第12版 下記を変更しました。 ■ 「 *入力チェックプログラム* 」の説明を変更しました。 ■ 「 *後処理* 」で動作対象とならない処理に関する説明を変更しました。 ■ 以下のページの図について、翻訳に対応する形に変更しました。 　■ 「 *後処理* 」 　■ 「 *前処理* 」 ■ 「 *JavaEE開発モデル* 」の以下の表を変更しました。 　■ 「 *ファイル取得処理の実装* 」の戻り値 　■ 「 *ファイル情報取得処理の実装* 」の戻り値 ■ 「 *スクリプト開発モデル* 」の以下の表を変更しました。 　■ 「 *ファイル取得処理の実装* 」の戻り値 　■ 「 *ファイル情報取得処理の実装* 」の戻り値
2018-04-01	第13版 下記を変更しました。 ■ 以下のページに画面アイテム「グリッドテーブル」に関する記載を追加しました。 　■ 「 *後処理* 」 　■ 「 *前処理* 」 　■ 「 *入力チェックプログラム* 」 　■ 「 *採番プログラム* 」 　■ 「 *登録データ情報管理API* 」
2019-04-01	第14版 下記を変更しました。 ■ 以下のページに画面アイテム「スプレッドシート」に関する記載を追加しました。 　■ 「 *登録情報マップ* 」

変更年月日	変更内容
2019-08-01	第15版 下記を変更・追加しました。 ■ 画面アイテム表示タイプを考慮した後処理プログラム内でのデータ利用方法に関する記載を追加しました。 ■ 「*後処理*（sendParamマップ）」 ■ 「*後処理*（sendParamオブジェクト）」 ■ 「ロジックフロー」の記載を追加しました。 ■ 「*後処理*」 ■ 「*前処理*」 ■ 「*入力チェックプログラム*」 ■ アプリケーション種別の列挙型に関する記載誤りを修正しました。 ■ 「*前処理*（PreProcessParameter）」 ■ 「*後処理*（PostProcessParameter）」 ■ 案件の再利用に関するパラメータの記載を追加しました。 ■ 「*前処理*（PreProcessParameter）」 ■ 「*前処理*（formaParamオブジェクト）」 ■ 入力チェックエラー情報設定メソッドについての記載を追加しました。 ■ 「*入力チェックプログラム*（addValidationError）」 ■ 「*入力チェックプログラム*（addValidationErrorForGridtable）」
2023-10-01	第16版 下記を変更・追加しました。 ■ 「*入力チェックプログラム*」の説明を変更しました。 ■ 以下のページに「wkhtmltopdf を利用したPDF出力」に関する説明を追加しました。 ■ 「*PDF出力処理*」 ■ 「*後処理プログラムでの実装手順*」 ■ 「*ジョブスケジューラ・非同期処理機能での実装手順*」
2025-10-01	第17版 下記を変更しました。 ■ 以下のページに「wkhtmltopdf を利用したPDF出力」に関する説明を修正しました。 ■ 「*PDF出力処理*」 ■ 「*後処理プログラムでの実装手順*」 ■ 「*ジョブスケジューラ・非同期処理機能での実装手順*」

はじめに

本書の目的

本書ではIM-FormaDesigner for Accel Platform で利用できるユーザプログラムを開発する場合の基本的な方法や注意点等について説明します。

対象読者

以下の条件を満たす開発者を対象としています。

- IM-FormaDesigner for Accel Platform で作成したアプリケーションに入力チェックを実装する。
- IM-FormaDesigner for Accel Platform で作成したアプリケーションの特定の処理後に独自の処理を組み込む。
- 以下のいずれかを理解していることが必須です。
  - JavaEE開発モデル（Java）
  - スクリプト開発モデル（サーバサイドJavaScript）

本書の構成

- *概要*
  IM-FormaDesigner for Accel Platform で利用できるユーザプログラムを開発するときの基本的な手順について説明します。
- *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。
- *前処理*
  前処理プログラムを実装する方法について説明します。
- *採番プログラム*
  採番プログラムを実装する方法について説明します。
- *登録データ情報管理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。
- *PDF出力処理*
  API を使用して参照画面をpdfに出力する方法について説明します。
- *テンプレートの格納先*
  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。

概要

IM-FormaDesigner for Accel Platform で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
- 前処理プログラム
- 採番プログラム

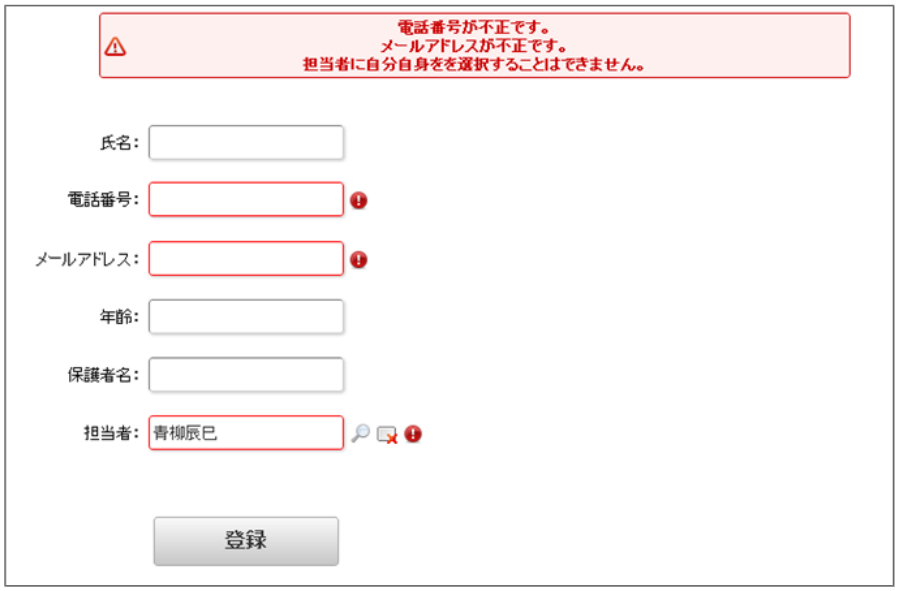
入力チェックプログラム

IM-FormaDesigner for Accel Platform では、入力チェックプログラムを実装することにより、次に挙げる例のような処理ができます。

- 画面アイテムの入力値に応じて、動的に入力チェックを実行する。
- 画面アイテムの入力値に対し、アプリケーション独自のチェックを行い、画面にエラーメッセージを表示する。

入力チェックの場合、１つのアプリケーション履歴に対して１つのプログラムを指定できます。
入力チェックプログラムとして、スクリプト開発モデル、またはロジックフローで作成したプログラムを利用できます。

- ユーザプログラムで入力チェックを行っている画面の例



後処理プログラム

IM-FormaDesigner for Accel Platform では、後処理プログラムを実装することにより、次に挙げる例のような処理ができます。

- アプリケーションデータの登録・更新・削除の処理後に、同一トランザクション内で独自の処理を実行する。

後処理の場合、１つのアプリケーション履歴に対して複数のプログラムを指定できます。
後処理プログラムとして、JavaEE開発モデル、スクリプト開発モデル、またはロジックフローで作成したプログラムを利用できます。
複数の後処理プログラムを実装した場合には、登録した順番に実行します。

前処理プログラム

IM-FormaDesigner for Accel Platform では、前処理プログラムを実装することにより、次に挙げる例のような処理ができます。

- 各画面を表示するタイミングに任意のプログラムを実行する。
- 各画面アイテムに初期表示する値を指定する。

前処理の場合、１つのアプリケーション履歴に対して複数のプログラムを指定できます。
前処理プログラムとして、JavaEE開発モデル、スクリプト開発モデル、またはロジックフローで作成したプログラムを利用できます。
複数の前処理プログラムを実装した場合には、登録した順番に実行します。

採番プログラム

IM-FormaDesigner for Accel Platform では、採番プログラムを実装することにより、次に挙げる例のような処理ができます。

- ある業務ルールに基づき「番号接頭語」を任意のプログラムにより作成する。
- 画面の入力内容に応じて動的に「番号接頭語」を変更する。

採番プログラムとして、JavaEE開発モデル、またはスクリプト開発モデルで作成したプログラムを利用できます。

コラム

「番号接頭語」を変更した場合は、同一の採番ルール定義の中でも「番号接頭語」ごとに採番の体系が異なります。

コラム

アプリケーションにユーザプログラムを登録する方法は、「 IM-FormaDesigner 作成者操作ガイド 」 を参照してください。

入力チェックプログラム

IM-FormaDesigner では、一時保存や申請、承認時にアプリケーションに入力された内容のチェックを任意のプログラムで実行できる機能を提供しています。
本項の内容に基づいて実装したユーザプログラムを IM-FormaDesigner のアプリケーションに設定する方法については、以下のドキュメントを参照してください。

- 「 IM-FormaDesigner 作成者操作ガイド 」-「 アプリケーションでユーザプログラムを利用する 」

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３つのパターンの実装が可能です。
ロジックフローの場合「 *アプリケーション固有の入力チェックをする。* 」と「 *エラーメッセージ表示順を指定する。* 」が実装可能です。

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」画面のプロパティで画面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画面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。

例）

- ある画面アイテムの入力値によって、他の画面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。
- 画面アイテムに設定した入力チェックを行うタイミングを制御したい。

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。
- 画面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。

エラーメッセージ表示順を指定する。

この場合は、画面に表示されるエラーメッセージの表示順序を独自に指定する必要があります。

入力チェックプログラムの実装

スクリプト開発モデル

スクリプト開発モデルにおいて、入力チェックプログラムを実装する手順を示します。

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。

- 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, resultObj)メソッド
- getErrorDisplayOrder(formaParam) メソッド

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。

formaParamオブジェクト

IM-FormaDesigner の画面の基本情報を保持します。

プロパティの概要

項目	説明
loginUserCd(String)	ログインユーザコード
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（アプリケーション履歴番号）
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 IM-BIS の場合、アプリケーション種別は BisApplication オブジェクトに定義されています。 - **BIS_APPLICATION_TYPE_WORKFLOW** ワークフロー - **BIS_APPLICATION_TYPE_BUSINESSFLOW** BISフロー
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）
temporaryFlag(String)	一時保存フラグ 申請・再申請・承認のいずれの場合も、一時保存が行われた際にはフラグを”1”に更新します。 - 一時保存時：1 - 登録時：0
processKey(String)	プロセスキー

コラム

一時保存時の入力チェックプログラムの実行

画面アイテム「ボタン（一時保存）」のプロパティで「入力チェック」を「しない」としている場合もユーザプログラムの入力チェックプログラムは実行されます。
一時保存時に入力チェックプログラムを実行させないようにするには、上記のformaParamオブジェクトの「temporaryFlag」を条件とした分岐を実装してください。

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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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」を参照してください。

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*データの取得メソッド*を参照してください。
但し、送信パラメータまたは API の取得データには、画面アイテム「ファイルアップロード」のアップロードファイル情報は含まれません。
また画面アイテム「採番」で採番方法を登録処理毎に設定した場合、採番処理は入力チェック後に行われるため送信パラメータに採番番号は含まれません。

- 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	システムタイムゾーンにおける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： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。

- 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルの取得値
  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。
- 画面アイテム表示タイプを考慮した入力チェックプログラムプログラム内でのデータ利用方法
  画面アイテム表示タイプが「入力可」は送信パラメータに含まれている為、上記をご利用ください。
  画面アイテム表示タイプが「参照」であるアイテムの値を利用する場合は「 IM-FormaDesigner プログラミングガイド 」-「 *データの取得メソッド*」を参照してください。
  ※表示タイプ「参照」は以下の問題がある為、送信パラメータに含まれません。
  ・ 実際に画面入力した値とDB登録済みの値が判断できません。
  ・ 参照の値は入力チェックを通さない為、改ざんされる可能性があります。

コラム

表示中画面以外の入力アイテム情報の取得について

複数のフォームを使用した画面を実行中、表示中画面以外のフォーム入力アイテム情報は、sendParamオブジェクトの「imfr_other_form_param」から取得可能です。
このプロパティは、表示された画面の入力アイテム情報を保持する為、一度も表示されていない画面の入力アイテム情報は取得できません。
フィールド値DB登録がONに設定されたアイテムのみ情報を取得できます。

PC表示/スマートフォン表示で処理を分岐する

画面表示方法ごとに入力チェックプログラムのロジックを変更したい場合は、実行中の画面表示タイプを取得し、処理を分岐します。

- 実行中の画面表示タイプは、sendParamオブジェクトのプロパティ imfr_display_client_typeから取得可能です。
  - PC表示
    - imfr_display_client_typeの値がpcです。
  - スマートフォン表示
    - imfr_display_client_typeの値がspです。

画面操作で処理を分岐する

画面操作方法ごとに入力チェックプログラムのロジックを変更したい場合は、画面操作時の遷移モードを取得し、処理を分岐します。

- 画面操作時の遷移モードは、sendParamオブジェクトのプロパティ imfr_transition_modeから取得可能です。
  - フォーム遷移方法「タブ切替」

画面操作	取得値
ボタン（次へ）	tab_contents
ボタン（戻る）	tab_contents
タブ選択	tab_contents
ボタン（一時保存）	×
ボタン（登録）	×
ボタン（BISフロー登録）	×

  - フォーム遷移方法「画面遷移」

画面操作	取得値
ボタン（次へ）	next
ボタン（戻る）	×（※）
ボタン（一時保存）	×
ボタン（登録）	×
ボタン（BISフロー登録）	×

※ フォーム遷移方法「画面遷移」の場合、アイテム「ボタン（戻る）」操作時は、入力チェックプログラムは実行されません。

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画面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。

パラメータ

formaParam、sendParam についての詳細は*共通パラメータ*を参照してください。

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

プロパティ詳細		
error (Boolean)	エラーフラグ 処理中にエラーが発生した場合はtrue、ない場合はfalseを設定します。	
errorMessage (String)	エラーメッセージ	
data (Object)	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。	
	%フィールド識別ID1% (String)	入力プロパティオブジェクト (Object)
		%プロパティ1% （下記の *入力チェックプロパティ一覧* 参照） %プロパティ2% ・・・・・・・・・・・・・・・・・・・ %プロパティ*n*%
	%フィールド識別ID2%	入力プロパティオブジェクト
	%フィールド識別ID3%	入力プロパティオブジェクト

※ 画面アイテム「ファイルアップロード」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*共通処理 ImFormaUtilオブジェクト*を参照してください。

入力チェックプロパティ一覧

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	最小入力値	Number
FRItems.PROP_PERMIT_MINUS	負数入力許可	Boolean
FRItems.PROP_PERMIT_DECIMAL	小数入力許可	Boolean
FRItems.PROP_DECIMAL_SIZE	小数部最大入力桁数	Number
FRItems.PROP_MIN_FILE_AMOUNT	最小ファイル数	Number
FRItems.PROP_MAX_FILE_AMOUNT	最大入力ファイル数	Number

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: {}
    };

    // 'telnumber' 英数字のみをfalse
    customPropObj['telnumber'][FRItems.PROP_ONLY_ASCII] = false;

    // 'age' 最大入力値を40 に設定
    customPropObj['age'][FRItems.PROP_MAX_LENGTH] = 40;

    // 'startdate' 必須をfalse
    customPropObj['startdate'][FRItems.PROP_REQUIRED] = false;

    // ' 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
    var fileId = ImFormaUtil.getFileUploadId(0);
    customPropObj[fileId] = {};
    // ファイルアップロード 最小入力ファイル数を 0
    customPropObj[fileId][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 };
}
```

> **コラム**
>
> 入力チェックプロパティは、画面アイテムプロパティで設定した入力チェックの内容を変更する場合に利用します。
> 画面アイテムプロパティで入力チェックが設定できないアイテムには利用できません。

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。

パラメータ

- formaParam、sendParam についての詳細は*共通パラメータ*を参照してください。
- 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、ない場合は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）が発生した場合はtrue、ない場合は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配列オブジェクト(Object)

checkid	入力チェックID
errorMessage	エラーメッセージ

プロパティ詳細	
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

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。
  - エラーがあった画面アイテムに対しエラー通知表示をする。
    （画面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画面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*共通処理 ImFormaUtilオブジェクト*を参照してください。

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 **********');

    var errorMessage = '';
    // 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 (checkOneValue(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 (checkTwoValues(sendParam['sample_field_2']), (sendParam['sample_field_3'])) {
        errorMessage = 'フィールド２の値が○○○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 tb1）内のフィールドに対するチェック
    var tableId = 'tb1';
    // 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
        var inputId;
        for (inputId in tableData[i]) {
            var checkValue = tableData[i][inputId];
            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 (inputId == 'tb1_textbox1') {
                if (checkCol1(checkValue)) {
                    errorMessage = 'XXXXXXX';
                    ImFormaUtil.addValidationError(resultObj, errorMessage, 'check3', [inputId], 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 (inputId == 'tb1_textbox2') {
                if (checkCol2(checkValue)) {
                    errorMessage = 'XXXXXXX';
                    ImFormaUtil.addValidationError(resultObj, errorMessage, 'check4', [inputId], i);
                }
            }
        }
    }
    // 返却オブジェクトを返却
    return resultObj;
}

function checkOneValue(inputValue) {
    // 「sample_field_1」に対する入力チェック処理を実装する。
}

function checkTwoValues(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 checkCol1(inputValue) {
    // 明細テーブル内フィールド「tb1_textbox1」に対する入力チェック処理を実装する。
}

function checkCol2(inputValue) {
    // 明細テーブル内フィールド「tb1_textbox2」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

> **コラム**
> 入力チェックIDは、入力チェックエラー情報設定メソッドの利用毎に、一意である必要があります。
> 入力チェックIDが重複していた場合、エラーメッセージが表示されない場合があります。

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。

エラーメッセージ表示順を指定する

- getErrorDisplayOrder(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画面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. 画面アイテムの重なり順（「フォーム・デザイナ」画面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

  なお、このメソッドにより複数入力フィールドを持つ画面アイテムの入力フィールドに対する画面アイテム内でのエラーメッセージの順序は、画面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）
  画面アイテム「期間」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]の順に変更できません。

パラメータ

formaParam についての詳細は*共通パラメータ*を参照してください。

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getItemIdById メソッド（ IM-FormaDesigner for Accel Platform 2013 Summer(Damask) 以前）、getItemIdById4Proc メソッド（ IM-FormaDesigner for Accel Platform 2013 Winter(Felicia) 以降）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画面アイテムの場合は、当該メソッドに任意の一つのフィールド識別ID を設定してください。
  画面アイテム「ファイルアップロード」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。

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」
    //  【◎】IM-FormaDesigner for Accel Platform 2013 Summer (8.0.4)以前
    // arry.push(ImFormaUtil.getItemIdById('sample_field_1'));
    // arry.push(ImFormaUtil.getItemIdById('sample_field_3'));

    //  【◎】IM-FormaDesigner for Accel Platform 2013 Winter (8.0.5)以降　getItemIdByIdは非推奨です。
    arry.push(ImFormaUtil.getItemIdById4Proc('sample_field_1', formaParam.processKey));
    arry.push(ImFormaUtil.getItemIdById4Proc('sample_field_3', formaParam.processKey));

    //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;
}
```

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できません。

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。
  エラーがあった画面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（画面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画面アイテムに依存）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）は画面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画面アイテムです。
  独自に作成した画面アイテムに対し、エラー通知表示を行いたい場合は、後述の*追加した画面アイテムにエラー通知表示を行う方法*を参照してください。

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)	エラー対象がテーブルの場合、エラーになった行のインデックスを設定します。 テーブル以外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。

> **コラム**
>
> 入力チェックIDは、入力チェックエラー情報設定メソッドの利用毎に、一意である必要があります。
> 入力チェックIDが重複していた場合、エラーメッセージが表示されない場合があります。

> **コラム**
>
> このメソッドは IM-FormaDesigner for Accel Platform 2019 Summer(Waltz) より、画面アイテム「グリッドテーブル」に対応しました。
> 画面アイテム「明細テーブル」と同様に使うことができます。

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。
  エラーがあった画面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。

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 (String)	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

> **コラム**
>
> このメソッドは IM-FormaDesigner for Accel Platform 2013 Winter(Felicia) 以降は非推奨です。
> 代わりにgetItemIdById4Procメソッドを使用してください。

- String getItemIdById4Proc (inputId, processKey)
  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。
  IM-FormaDesigner for Accel Platform 2013 Winter(Felicia) 以降から利用できます。

パラメータ

inputId (String)	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。
processKey (String)	formaParamオブジェクトのプロセスキー（formaParam.processKey）を指定します。

返却値

アイテムID

ファイルアップロードID取得メソッド

- String getFileUploadId (index)
  画面アイテム「ファイルアップロード」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画面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 (Number)	index にはフォーム内に出現する画面アイテム「ファイルアップロード」の順番を指定します。 一番目の画面アイテム「ファイルアップロード」の index には「0」を指定します。 ※「フォーム内に出現する画面アイテム「ファイルアップロード」の順番」とは、画面アイテムの重なり順（「フォーム・デザイナ」画面のラベル一覧の表示順）を意味します。

  返却値

ファイルアップロードID

追加した画面アイテムにエラー通知表示を行う方法

追加した画面アイテムが 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できません。

※独自に追加した画面アイテムでも画面アイテム「文字列」など、標準で提供している画面アイテムと同じエラー通知処理を行っている場合は addValidationError メソッドを使用できます。

独自のエラー通知処理を行っている画面アイテムに対し、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。

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

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、ない場合は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）が発生した場合はtrue、ない場合は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配列オブジェクト(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

- 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。

共通処理 ImBisFormaUtilオブジェクト

ImBisFormaUtil オブジェクトは、 IM-BIS を導入している環境で、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 IM-BIS を導入している環境の入力チェックプログラム内でしか使用できません。

入力チェックエラー情報設定メソッド

- void addValidationErrorForGridtable(resultObj, errorMessage, sendParam, 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。
  エラーがあった画面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を行う場合にこのメソッドを利用します。
  このメソッドは画面アイテム「グリッドテーブル」専用の入力チェックエラー表示メソッドです。

  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
sendParam (Object)	送信パラメータオブジェクト validateメソッドの引数で受け取った送信パラメータオブジェクトを設定します。
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 エラー対象として設定可能な列タイプ（文字列/数値/日付/関数/一覧選択/セレクトボックス）のフィールド識別IDを設定します。 複数設定する場合、同一のグリッドテーブルのフィールド識別IDのみ設定できます。
index (Number)	エラーになった行のインデックスを設定します。

  返却値

返却値はありません。

> **コラム**
> このメソッドは IM-FormaDesigner for Accel Platform 2019 Summer(Waltz) 以降は非推奨です。
> 代わりに *共通処理 ImFormaUtilオブジェクト* のaddValidationErrorメソッドを使用してください。

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：グリッドテーブル **********');

    var errorMessage = '';
    // グリッドテーブル（テーブル識別ID gt1）内のフィールドに対するチェック
    var gridtableId = 'gt1';
    // グリッドテーブルデータの取得
    var gridtableData = sendParam[gridtableId];

    // 入力行数分チェック
    for (var i = 0, length = gridtableData.length; i < length; i++) {
        // 列数分チェック
        var inputId;
        for (inputId in gridtableData[i]) {
            var checkValue = gridtableData[i][inputId];
            if (checkValue == '') continue;
            //入力チェック(5) 列1のフィールドに対するチェック
            if (inputId == 'gt1_textbox1') {
                if (checkCol3(checkValue)) {
                    errorMessage = 'XXXXXXX';
                    ImBisFormaUtil.addValidationErrorForGridtable(resultObj, errorMessage, sendParam, 'check5', [inputId], i);
                }
                //入力チェック(6) 列2のフィールドに対するチェック
            } else if (inputId == 'gt1_textbox2') {
                if (checkCol4(checkValue)) {
                    errorMessage = 'XXXXXXX';
                    ImBisFormaUtil.addValidationErrorForGridtable(resultObj, errorMessage, sendParam, 'check6', [inputId], i);
                }
            } else if (.......) {
                .......
            }
        }
    }
    // 返却オブジェクトを返却
    return resultObj;
}

function checkCol3(inputValue) {
    //グリッドテーブル内フィールド「gt1_textbox1」に対する入力チェック処理を実装する。
}

function checkCol4(inputValue) {
    //グリッドテーブル内フィールド「gt1_textbox2」に対する入力チェック処理を実装する。
}
```

ロジックフロー

ロジックフローにおいて、入力チェックプログラムを実装する手順を示します。

IM-LogicDesignerの操作方法につきましては「IM-LogicDesigner ユーザ操作ガイド」を参照してください。

入力値

ロジックフローの入力値に渡される入力パラメータ情報です。

```
formaParam <object>
 ├─ loginUserCd <string>
 ├─ applicationId <string>
 ├─ applicationNo <long>
 ├─ insertId <string>
 ├─ applicationType <string>
 ├─ appPageType <string>
 ├─ temporaryFlag <string>
 └─ processKey <string>
sendParam <object>
workflowParameter <object>
 ├─ imwApplyBaseDate <string>
 ├─ imwArriveType <string>
 ├─ imwAuthUserCode <string>
 ├─ imwCallOriginalPagePath <string>
 ├─ imwCallOriginalParams <string>
 ├─ imwContentsId <string>
 ├─ imwContentsVersionId <string>
 ├─ imwFlowId <string>
 ├─ imwFlowVersionId <string>
 ├─ imwNodeId <string>
 ├─ imwPageType <string>
 ├─ imwRouteId <string>
 ├─ imwRouteVersionId <string>
 ├─ imwSerialProcParams <string>
 ├─ imwSystemMatterId <string>
 ├─ imwUserCode <string>
 └─ imwUserDataId <string>
imbpmExecutionInfo <object>
 ├─ id <string>
 ├─ processInstanceId <string>
 ├─ businessKey <string>
 ├─ processDefinitionId <string>
 ├─ superExecutionId <string>
 ├─ currentActivityId <string>
 ├─ currentActivityName <string>
 ├─ variablesLocal <object>
 └─ variables <object>
```

入力値	説明
formaParam<object>	*formaParamオブジェクト* を参照
sendParam<object>	*sendParamオブジェクト* を参照
workflowParameter<object>	*workflowParameterオブジェクト* を参照
imbpmExecutionInfo<object>	*imbpmExecutionInfoオブジェクト* を参照

コラム

IM-LogicDesignerの「ロジックフロー定義編集」入出力設定では、JSON入力機能により以下のJSON文字列を取り込むことができます。

```
{
 "formaParam": {
  "loginUserCd": "",
  "applicationId": "",
  "applicationNo": 0,
  "insertId": "",
  "applicationType": "",
  "appPageType": "",
  "temporaryFlag": "",
  "processKey": ""
 },
 "sendParam": {},
 "workflowParameter": {
  "imwApplyBaseDate": "",
  "imwArriveType": "",
  "imwAuthUserCode": "",
  "imwCallOriginalPagePath": "",
  "imwCallOriginalParams": "",
  "imwContentsId": "",
  "imwContentsVersionId": "",
  "imwFlowId": "",
  "imwFlowVersionId": "",
  "imwNodeId": "",
  "imwPageType": "",
  "imwRouteId": "",
  "imwRouteVersionId": "",
  "imwSerialProcParams": "",
  "imwSystemMatterId": "",
  "imwUserCode": "",
  "imwUserDataId": ""
 },
 "imbpmExecutionInfo": {
  "id": "",
  "processInstanceId": "",
  "businessKey": "",
  "processDefinitionId": "",
  "superExecutionId": "",
  "currentActivityId": "",
  "currentActivityName": "",
  "variablesLocal": {},
  "variables": {}
 }
}
```

formaParamオブジェクト

IM-FormaDesigner の画面の基本情報を保持します。

入力値	説明
loginUserCd<string>	ログインユーザコード
applicationId<string>	アプリケーションID
applicationNo<long>	アプリケーションバージョンNO（アプリケーション履歴番号）
insertId<string>	データ登録ID

入力値	説明
applicationType<string>	アプリケーション種別 ▪ **std** 標準 ▪ **imw** IM-Workflow IM-BIS の場合 ▪ **bis_wkf** ワークフロー ▪ **bis_bf** BISフロー
appPageType<string>	アプリケーションページ種別 ▪ **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）
temporaryFlag<string>	一時保存フラグ 申請・再申請・承認のいずれの場合も、一時保存が行われた際にはフラグを”1”に更新します。 ▪ 一時保存時：1 ▪ 登録時：0
processKey<string>	プロセスキー

コラム

一時保存時の入力チェックプログラムの実行

画面アイテム「ボタン（一時保存）」のプロパティで「入力チェック」を「しない」としている場合もユーザプログラムの入力チェックプログラムは実行されます。
一時保存時に入力チェックプログラムを実行させないようにするには、上記のformaParamオブジェクトの「temporaryFlag」を条件とした分岐を実装してください。

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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルのパラメータ名はテーブル識別ID です。

画面アイテム「ファイルアップロード」のアップロードファイル情報は含まれません。
また画面アイテム「採番」で採番方法を登録処理毎に設定した場合、採番処理は入力チェック後に行われるため送信パラメータに採番番号は含まれません。

- 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

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	<string>	▪ 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 <string>	▪ 数値 ▪ （関数※1） ▪ （一覧選択※1）
日付またはタイムスタンプ	システムタイムゾーンにおける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の <string>	▪ 日付 ▪ 期間 ▪ （関数※1） ▪ （一覧選択※1）

※1： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。

- 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルの取得値
  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 <object[]> です。
  各列のデータ型は列タイプに依存します。
- 画面アイテム表示タイプを考慮した入力チェックプログラムプログラム内でのデータ利用方法
  画面アイテム表示タイプが「入力可」は送信パラメータに含まれている為、上記をご利用ください。
  ※表示タイプ「参照」は以下の問題がある為、送信パラメータに含まれません。
  ・ 実際に画面入力した値とDB登録済みの値が判断できません。
  ・ 参照の値は入力チェックを通さない為、改ざんされる可能性があります。

> コラム
>
> **表示中画面以外の入力アイテム情報の取得について**
>
> 複数のフォームを使用した画面を実行中、表示中画面以外のフォーム入力アイテム情報は、sendParamオブジェクトの「imfr_other_form_param」から取得可能です。
> このプロパティは、表示された画面の入力アイテム情報を保持する為、一度も表示されていない画面の入力アイテム情報は取得できません。
> フィールド値DB登録がONに設定されたアイテムのみ情報を取得できます。

PC表示/スマートフォン表示で処理を分岐する

画面表示方法ごとに入力チェックプログラムのロジックを変更したい場合は、実行中の画面表示タイプを取得し、処理を分岐します。

- 実行中の画面表示タイプは、sendParamオブジェクトのプロパティ imfr_display_client_typeから取得可能です。
  - PC表示
    - imfr_display_client_typeの値がpcです。
  - スマートフォン表示
    - imfr_display_client_typeの値がspです。

画面操作で処理を分岐する

画面操作方法ごとに入力チェックプログラムのロジックを変更したい場合は、画面操作時の遷移モードを取得し、処理を分岐します。

- 画面操作時の遷移モードは、sendParamオブジェクトのプロパティ imfr_transition_modeから取得可能です。
  - フォーム遷移方法「タブ切替」

画面操作	取得値
ボタン（次へ）	tab_contents

画面操作	取得値
ボタン（戻る）	tab_contents
タブ選択	tab_contents
ボタン（一時保存）	×
ボタン（登録）	×
ボタン（BISフロー登録）	×

- フォーム遷移方法「画面遷移」

画面操作	取得値
ボタン（次へ）	next
ボタン（戻る）	×（※）
ボタン（一時保存）	×
ボタン（登録）	×
ボタン（BISフロー登録）	×

※ フォーム遷移方法「画面遷移」の場合、アイテム「ボタン（戻る）」操作時は、入力チェックプログラムは実行されません。

workflowParameterオブジェクト

ワークフローに関するパラメータを保持します。
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が利用できます。

各画面で取得が可能な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 」-「 リクエストパラメータ 」を参照してください。

入力値	説明
imwApplyBaseDate<string>	申請基準日
imwArriveType<string>	到達種別
imwAuthUserCode<string>	権限者コード
imwCallOriginalPagePath<string>	呼び出し元ページパス
imwCallOriginalParams<string>	呼び出し元パラメータ
imwContentsId<string>	コンテンツID
imwContentsVersionId<string>	コンテンツバージョンID
imwFlowId<string>	フローID
imwFlowVersionId<string>	フローバージョンID
imwNodeId<string>	ノードID
imwPageType<string>	画面種別
imwRouteId<string>	ルートID
imwRouteVersionId<string>	ルートバージョンID
imwSerialProcParams<string>	連続処理パラメータ
imwSystemMatterId<string>	システム案件ID
imwUserCode<string>	処理者コード
imwUserDataId<string>	ユーザデータID

imbpmExecutionInfoオブジェクト

案件の連携元タスクのエグゼキューション情報を保持します。
IM-BPM for Accel Platformの申請タスクまたは起票タスクからIM-Workflowが実行された場合利用できます。

詳細については、「IM-BPM プロセスデザイナ 操作ガイド」 -「申請タスク」または 「起票タスク」を参照してください。

入力値	説明
id<string>	エグゼキューションID
processInstanceId<string>	プロセスインスタンスID
businessKey<string>	業務キー
processDefinitionId<string>	プロセス定義ID
superExecutionId<string>	親プロセスのエグゼキューションID
currentActivityId<string>	タスクのアクティビティID
currentActivityName<string>	タスクのアクティビティ名
variablesLocal<object>	変数（スコープがエグゼキューションであるもの） ・ 変数名をキー、変数値を値として持つ object です。
variables<object>	変数（スコープがプロセスインスタンスであるもの） ・ 変数名をキー、変数値を値として持つ object です。

出力値

ロジックフローの出力値に設定する情報です。

```
error <boolean>
errorMessage <string>
errorItems <object[]>
 ├─ inputId <string[]>
 ├─ errorMessage <string>
 ├─ localizedErrorMessages <object>
 │   ├─ ja <string>
 │   ├─ en <string>
 │   └─ zh_CN <string>
 └─ index <integer>
```

説明

画面アイテムの入力チェック

- 入力チェックエラーがなかった場合
  errorItems<object[]> に何も設定しません。
- 入力チェックエラーがあった場合
  errorItems<object[]> 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。
    （画面に表示される入力チェックエラーメッセージの順番は配列の先頭より昇順です）
  - エラーがあった画面アイテムに対しエラー通知表示をする。
    （画面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画面アイテムに依存します）

処理エラー時の動作

- 処理中に入力チェック以外のエラーが発生した場合は、出力値のエラーフラグ error<boolean> に true を設定してください。
- エラーフラグが true の場合は、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、例外ログが出力されます。

プロパティ

出力値	必須/任意	説明
error<boolean>	必須	エラーフラグ true を設定した場合、当処理をエラー終了します。 処理が正常に終了した場合は false を設定してください。
errorMessage<string>	任意	設定したメッセージを例外ログとして出力します。

出力値	必須/任意	説明
errorItems<object[]>	任意	入力チェックエラーアイテム情報です。 入力チェックを行う場合は必ずこのパラメータを定義してください。 画面に表示される入力チェックエラーメッセージの順番は配列の先頭より昇順です。
- inputId<string[]>	任意	エラーになったアイテムの識別ID配列です。 フォームにあるアイテムのフィールド識別IDを設定します。 画面アイテム「ファイルアップロード」はアイテム識別IDを設定します。 一つのエラーメッセージに対し、複数のフィールドが紐づく場合は、配列に識別IDを全て設定します。 入力チェックエラーメッセージのみを表示させたい場合はこのプロパティを省略します。
- errorMessage<string>	任意	入力チェックエラーメッセージです。 設定したメッセージを画面に表示します。 ロケールごとにメッセージを用意する必要がなければこのプロパティを利用してください。
- localizedMessages<object>	任意	ロケール別入力チェックエラーメッセージです。 設定したメッセージをロケールに従って画面に表示します。 アカウントコンテキストから解決されたロケールに一致するメッセージを使用します。 入力チェックエラーアイテム情報の errorMessage が設定されている場合はこのプロパティは使用されません。
– ja<string>	任意	日本語ロケールメッセージ
– en<string>	任意	英語ロケールメッセージ
– zh_CN<string>	任意	中国語ロケールメッセージ
- index <integer>	任意	エラー行番号です。 入力チェック対象にテーブルが含まれる場合に定義します。

コラム

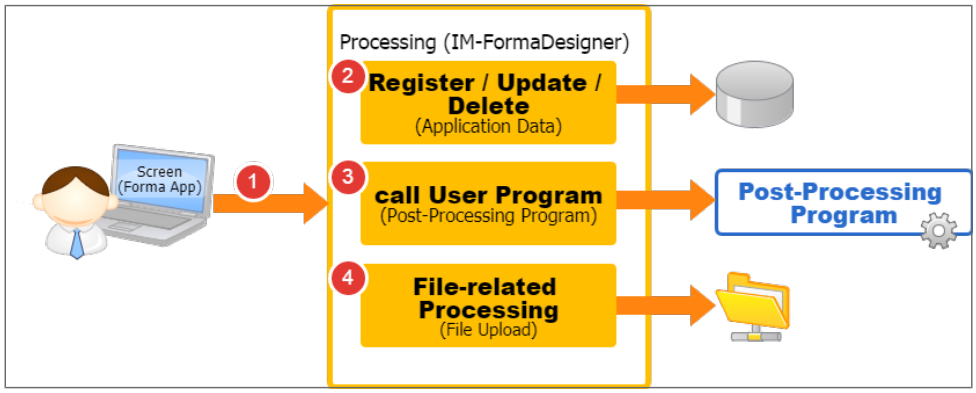
IM-LogicDesignerの「ロジックフロー定義編集」入出力設定では、JSON入力機能により以下のJSON文字列を取り込むことができます。

```
{
 "error": false,
 "errorMessage": "",
 "errorItems": [
  {
   "inputId": [
    ""
   ],
   "errorMessage": "",
   "localizedErrorMessages": {
    "ja": "",
    "en": "",
    "zh_CN": ""
   },
   "index": 0
  }
 ]
}
```

後処理

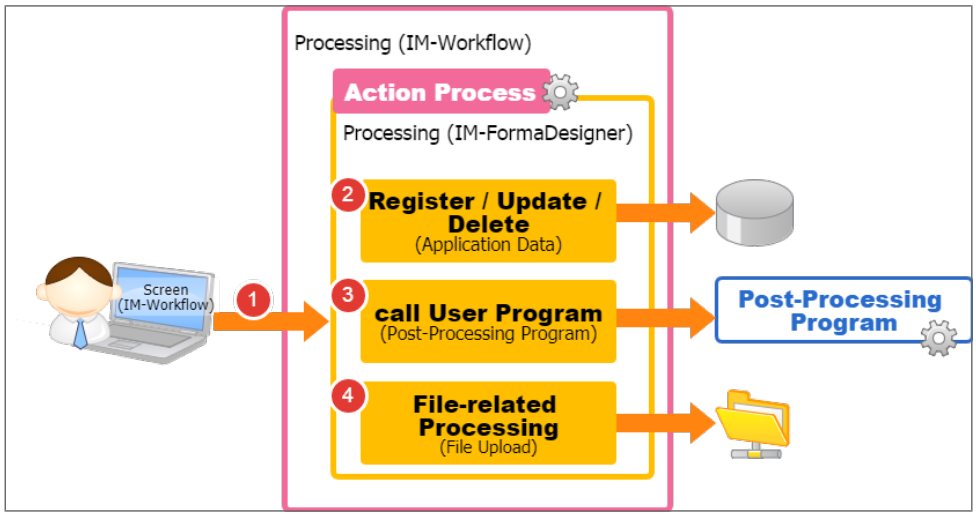
IM-FormaDesigner for Accel Platform では、アプリケーションデータの登録、更新、削除などデータの操作が行われた後に任意のプログラムを実行することができる機能を提供しています。
アプリケーションの種別を問わずデータの操作が行われたタイミング※で後処理は実行されます。
※ファイルアップロードに関する処理は後処理の実行後に行われます。
IM-Workflow の処理に合わせて後処理を実行したい場合には、 IM-Workflow のユーザプログラム機能をご利用ください。

- 後処理が呼ばれる流れ
  - **アプリケーション種別：標準の場合**



1. アプリケーションのデータに対する登録、更新、削除のいずれかの処理を実行します。
2. アプリケーションデータの登録・更新・削除処理が実行されます。
3. 後処理プログラムの呼び出しが行われ、登録されている後処理プログラムが実行されます。
4. ファイル関連（ファイルアップロード）の処理が実行されます。

  - **アプリケーション種別：IM-Workflowの場合**



1. ワークフローの申請・再申請・承認を実行します。
   承認については、「連続処理」も後処理プログラムのトリガ操作に含まれます。
   「一括処理」・「自動処理」による承認では、アプリケーションデータの操作が実施されないため、後処理プログラムは実行されません。
   後続の IM-FormaDesigner による処理は、 IM-Workflow のアクション処理内で実行されます。
2. アプリケーションデータの登録・更新・削除処理が実行されます。
3. 後処理プログラムの呼び出しが行われ、登録されている後処理プログラムが実行されます。
4. ファイル関連（ファイルアップロード）の処理が実行されます。

後処理プログラムの実装

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。

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
  - データの一時保存時
    preserve(PostProcessParameter formaParam, Map<String, Object> sendParam) メソッド

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。

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

IM-FormaDesigner の画面の基本情報を保持するクラスです。

メソッドの概要

メソッド	説明
String getLoginUserCd()	ログインユーザコードを返却します。
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（アプリケーション履歴番号）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 IM-BIS の場合、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.BisApplicationType ▪ **BIS_WKF** ワークフロー ▪ **BIS_BF** BISフロー

メソッド	説明
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.ApplicationPageType ■ **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）
String getProcessKey()	プロセスキーを返却します。

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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルを除く入力アイテムのキーはフィールド識別ID です。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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 」-「 リクエストパラメータ 」を参照してください。

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*データの取得メソッド*を参照してください。
但し、送信パラメータまたは API の取得データには、画面アイテム「ファイルアップロード」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	システムタイムゾーンにおける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： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。
- 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルの取得値

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 Map の List です。
各列のデータ型は列タイプに依存します。

- 画面アイテム表示タイプを考慮した後処理プログラム内でのデータ利用方法
  画面アイテム表示タイプが「入力可」は送信パラメータに含まれている為、上記をご利用ください。
  画面アイテム表示タイプが「参照」であるアイテムの値を利用する場合は「IM-FormaDesigner プログラミングガイド」-「*データの取得メソッド*」を参照してください。
  ※表示タイプ「参照」は以下の問題がある為、送信パラメータに含まれません。
  ・ 実際に画面入力した値とDB登録済みの値が判断できません。
  ・ 参照の値は入力チェックを通さない為、改ざんされる可能性があります。

PC表示/スマートフォン表示で処理を分岐する

画面表示方法ごとに後処理プログラムのロジックを変更したい場合は、実行中の画面表示タイプを取得し、処理を分岐します。

- 実行中の画面表示タイプは、sendParamマップのキー imfr_display_client_typeから取得可能です。
  - PC表示
    - imfr_display_client_typeの値はpcです。
  - スマートフォン表示
    - imfr_display_client_typeの値はspです。

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。
処理中にエラーが発生した場合に画面に任意のメッセージを表示したい場合は、FormaUserProcessException 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。

jp.co.intra_mart.foundation.forma.exception.FormaUserProcessException

ユーザプログラム実行時例外クラスです。

主なメソッドの概要

メソッド	説明
void setUserMessageId(final String userMessageId)	画面に表示するメッセージIDを指定します。
void setUserMessage(final String userMessage)	画面に表示するメッセージを指定します。userMessageIdが設定されている場合は、userMessageIdが適用されます。
void setOutputLog(final boolean isOutputLog)	ログ出力フラグを設定します。省略した場合はtrue（出力する）です。

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ください。

実装例

```java
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。

        // エラー発生時に画面に表示するメッセージを指定する場合は、FormaUserProcessException をthrowします。
        if (XXXX) {
            final FormaUserProcessException error = new FormaUserProcessException(new XXXXException());
            error.setUserMessage("----  画面に表示したいメッセージ----");
            // error.setOutputLog(false);
            throw error;
        }
    }

    /**
     * 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。
    }

    /**
     * 一時保存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 preserve(PostProcessParameter formaParam, Map<String, Object> sendParam) throws Exception {
        System.out.println("********** JAVAEE 開発：一時保存後処理サンプルプログラム**********");
        // ここに一時保存処理後に実行する処理を記述します。
    }
}
```

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。

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
- データの一時保存時
  preserve(formaParam, sendParam) メソッド

パラメータ

formaParamオブジェクト

IM-FormaDesigner の画面の基本情報を保持します。

項目	説明
loginUserCd(String)	ログインユーザコード
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（アプリケーション履歴番号）
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
	IM-BIS の場合、アプリケーション種別は BisApplication オブジェクトに定義されています。
	- **BIS_APPLICATION_TYPE_WORKFLOW**
	  ワークフロー
	- **BIS_APPLICATION_TYPE_BUSINESSFLOW**
	  BISフロー

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）

processKey(String)	プロセスキー

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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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」-「リクエストパラメータ」を参照してください。

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*データの取得メソッド*を参照してください。
但し、送信パラメータまたは API の取得データには、画面アイテム「ファイルアップロード」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	システムタイムゾーンにおける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： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。

- 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルの取得値
  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。
- 画面アイテム表示タイプを考慮した後処理プログラム内でのデータ利用方法
  画面アイテム表示タイプが「入力可」は送信パラメータに含まれている為、上記をご利用ください。
  画面アイテム表示タイプが「参照」であるアイテムの値を利用する場合は「IM-FormaDesigner プログラミングガイド」-「*データの取得メソッド*」を参照してください。
  ※表示タイプ「参照」は以下の問題がある為、送信パラメータに含まれません。
  ・ 実際に画面入力した値とDB登録済みの値が判断できません。
  ・ 参照の値は入力チェックを通さない為、改ざんされる可能性があります。

PC表示/スマートフォン表示で処理を分岐する

画面表示方法ごとに後処理プログラムのロジックを変更したい場合は、実行中の画面表示タイプを取得し、処理を分岐します。

- 実行中の画面表示タイプは、sendParamオブジェクトのキー imfr_display_client_typeから取得可能です。
  - PC表示
    - imfr_display_client_typeの値はpcです。
  - スマートフォン表示
    - imfr_display_client_typeの値はspです。

返却値とエラー処理

```
返却オブジェクト
 ├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。）
 ├userErrorMessageId：後処理プログラム実行中のエラー発生時に画面に表示するメッセージのID
 ├userErrorMessage：後処理プログラム実行中のエラー発生時に画面に表示するメッセージ（※userErrorMessageIdに設定がある場合はuserErrorMessageIdが適用されます。）
 └outputLog：ログ出力有無フラグ　ユーザプログラム実行中にエラーが発生した際のログ出力有無を設定します。　省略した場合はtrue（出力する）です。
```

エラー発生時には返却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。

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ください。

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'};
    // エラー発生時に画面に表示するメッセージを指定する場合
    // return {'error' : true, 'userErrorMessage' :'----  画面に表示したいメッセージ----'};
}

// 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
    };
}

// 一時保存
function preserve(formaParam, sendParam) {
    Debug.print('********** スクリプト開発：一時保存後処理サンプルプログラム **********');
    // ここに一時保存処理後に実行する処理を記述します。
    return {
        'error' : false
    };
}
```

ロジックフロー

ロジックフローにおいて、後処理プログラムを実装する手順を示します。

IM-LogicDesignerの操作方法につきましては「IM-LogicDesigner ユーザ操作ガイド」を参照してください。

入力値

ロジックフローの入力値に渡される入力パラメータ情報です。

```
executeProcessType <string>
postProcessParameter <object>
 ├─ loginUserCd <string>
 ├─ applicationId <string>
 ├─ applicationNo <long>
 ├─ insertId <string>
 ├─ applicationType <string>
 ├─ appPageType <string>
 └─ processKey <string>
sendParam <object>
postProcessWorkflowParameter <object>
 ├─ imwApplyBaseDate <string>
 ├─ imwArriveType <string>
 ├─ imwAuthUserCode <string>
 ├─ imwCallOriginalPagePath <string>
 ├─ imwCallOriginalParams <string>
 ├─ imwContentsId <string>
 ├─ imwContentsVersionId <string>
 ├─ imwFlowId <string>
 ├─ imwFlowVersionId <string>
 ├─ imwNodeId <string>
 ├─ imwPageType <string>
 ├─ imwRouteId <string>
 ├─ imwRouteVersionId <string>
 ├─ imwSerialProcParams <string>
 ├─ imwSystemMatterId <string>
 ├─ imwUserCode <string>
 └─ imwUserDataId <string>
imbpmExecutionInfo <object>
 ├─ id <string>
 ├─ processInstanceId <string>
 ├─ businessKey <string>
 ├─ processDefinitionId <string>
 ├─ superExecutionId <string>
 ├─ currentActivityId <string>
 ├─ currentActivityName <string>
 ├─ variablesLocal <object>
 └─ variables <object>
```

入力値	説明
executeProcessType<string>	*executeProcessType* を参照
postProcessParameter<object>	*postProcessParameterオブジェクト* を参照
sendParam<object>	*sendParamオブジェクト* を参照
postProcessWorkflowParameter<object>	*postProcessWorkflowParameterオブジェクト* を参照
imbpmExecutionInfo<object>	*imbpmExecutionInfoオブジェクト* を参照

コラム

IM-LogicDesignerの「ロジックフロー定義編集」入出力設定では、JSON入力機能により以下のJSON文字列を取り込むことができます。

```
{
 "executeProcessType": "",
 "postProcessParameter": {
  "loginUserCd": "",
  "applicationId": "",
  "applicationNo": 0,
  "insertId": "",
  "applicationType": "",
  "appPageType": "",
  "processKey": ""
 },
 "sendParam": {},
 "postProcessWorkflowParameter": {
  "imwApplyBaseDate": "",
  "imwArriveType": "",
  "imwAuthUserCode": "",
  "imwCallOriginalPagePath": "",
  "imwCallOriginalParams": "",
  "imwContentsId": "",
  "imwContentsVersionId": "",
  "imwFlowId": "",
  "imwFlowVersionId": "",
  "imwNodeId": "",
  "imwPageType": "",
  "imwRouteId": "",
  "imwRouteVersionId": "",
  "imwSerialProcParams": "",
  "imwSystemMatterId": "",
  "imwUserCode": "",
  "imwUserDataId": ""
 },
 "imbpmExecutionInfo": {
  "id": "",
  "processInstanceId": "",
  "businessKey": "",
  "processDefinitionId": "",
  "superExecutionId": "",
  "currentActivityId": "",
  "currentActivityName": "",
  "variablesLocal": {},
  "variables": {}
 }
}
```

executeProcessType

アプリケーション実行種別を保持します。
プログラムを実行するタイミングに合わせて以下の値が取得可能です。

- **regist**
  データの登録時
- **update**
  データの更新時
- **remove**
  データの削除時
- **preserve**
  データの一時保存時

postProcessParameterオブジェクト

IM-FormaDesigner の画面の基本情報を保持します。

項目	説明
loginUserCd<string>	ログインユーザコード

項目	説明
applicationId<string>	アプリケーションID
applicationNo<long>	アプリケーションバージョンNO（アプリケーション履歴番号）
insertId<string>	データ登録ID
applicationType<string>	アプリケーション種別 ▪ **std** 標準 ▪ **imw** IM-Workflow IM-BIS の場合 ▪ **bis_wkf** ワークフロー ▪ **bis_bf** BISフロー
appPageType<string>	アプリケーションページ種別 ▪ **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）
processKey<string>	プロセスキー

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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル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。
画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルのパラメータ名はテーブル識別ID です。

画面アイテム「ファイルアップロード」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	<string>	▪ 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 <string>	▪ 数値 ▪ （関数※1） ▪ （一覧選択※1）

アイテムのデータ型	取得値	該当する主なアイテム
日付またはタイムスタンプ	システムタイムゾーンにおける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の <string>	▪ 日付 ▪ 期間 ▪ （関数※1） ▪ （一覧選択※1）

※1： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。

- 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルの取得値
  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 <object[]> です。
  各列のデータ型は列タイプに依存します。
- 画面アイテム表示タイプを考慮した後処理プログラム内でのデータ利用方法
  画面アイテム表示タイプが「入力可」は送信パラメータに含まれている為、上記をご利用ください。
  ※表示タイプ「参照」は以下の問題がある為、送信パラメータに含まれません。
  ・ 実際に画面入力した値とDB登録済みの値が判断できません。
  ・ 参照の値は入力チェックを通さない為、改ざんされる可能性があります。

PC表示/スマートフォン表示で処理を分岐する

画面表示方法ごとに後処理プログラムのロジックを変更したい場合は、実行中の画面表示タイプを取得し、処理を分岐します。

- 実行中の画面表示タイプは、sendParamオブジェクトのキー imfr_display_client_typeから取得可能です。
  - PC表示
    - imfr_display_client_typeの値はpcです。
  - スマートフォン表示
    - imfr_display_client_typeの値はspです。

postProcessWorkflowParameterオブジェクト

ワークフローに関するパラメータを保持します。
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が利用できます。

各画面で取得が可能な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」-「リクエストパラメータ」を参照してください。

入力値	説明
imwApplyBaseDate<string>	申請基準日
imwArriveType<string>	到達種別
imwAuthUserCode<string>	権限者コード
imwCallOriginalPagePath<string>	呼び出し元ページパス
imwCallOriginalParams<string>	呼び出し元パラメータ
imwContentsId<string>	コンテンツID
imwContentsVersionId<string>	コンテンツバージョンID
imwFlowId<string>	フローID
imwFlowVersionId<string>	フローバージョンID
imwNodeId<string>	ノードID
imwPageType<string>	画面種別
imwRouteId<string>	ルートID
imwRouteVersionId<string>	ルートバージョンID
imwSerialProcParams<string>	連続処理パラメータ
imwSystemMatterId<string>	システム案件ID
imwUserCode<string>	処理者コード

入力値	説明
imwUserDataId<string>	ユーザデータID

imbpmExecutionInfoオブジェクト

案件の連携元タスクのエグゼキューション情報を保持します。
IM-BPM for Accel Platformの申請タスクまたは起票タスクからIM-Workflowが実行された場合利用できます。

詳細については、「IM-BPM プロセスデザイナ 操作ガイド」 - 「申請タスク」または 「起票タスク」を参照してください。

入力値	説明
id<string>	エグゼキューションID
processInstanceId<string>	プロセスインスタンスID
businessKey<string>	業務キー
processDefinitionId<string>	プロセス定義ID
superExecutionId<string>	親プロセスのエグゼキューションID
currentActivityId<string>	タスクのアクティビティID
currentActivityName<string>	タスクのアクティビティ名
variablesLocal<object>	変数（スコープがエグゼキューションであるもの） ・ 変数名をキー、変数値を値として持つ object です。
variables<object>	変数（スコープがプロセスインスタンスであるもの） ・ 変数名をキー、変数値を値として持つ object です。

出力値

ロジックフローの出力値に設定する情報です。

```
error <boolean>
errorMessage <string>
userErrorMessage <string>
localizedUserErrorMessages <object>
 ├ ja <string>
 ├ en <string>
 └ zh_CN <string>
outputLog <boolean>
```

説明

処理エラー時の動作

- 処理中にエラーが発生した場合は error<boolean> に true を設定してください。
- エラーフラグが true の場合は、画面上部にエラーメッセージを表示します。
  - 画面に表示するエラーメッセージの設定がなかった場合は、デフォルトのエラーメッセージが表示されます。
  - 例外ログの出力有無は outputLog<boolean> で設定できます。

プロパティ

出力値	必須/任意	説明
error<boolean>	必須	エラーフラグ true を設定した場合、当処理をエラー終了します。 処理が正常に終了した場合は false を設定してください。
errorMessage<string>	任意	設定したメッセージを例外ログとして出力します。
userErrorMessage<string>	任意	設定したメッセージを画面に表示します。 ロケールごとにメッセージを用意する必要がなければこのプロパティを利用してください。 このプロパティを省略した場合、画面にはデフォルトのエラーメッセージが表示されます。

出力値	必須/任意	説明
localizedUserErrorMessages<object>	任意	設定したメッセージをロケールに従って画面に表示します。 アカウントコンテキストから解決されたロケールに一致するメッセージを使用します。 userErrorMessage が設定されている場合はこのプロパティは使用されません。
- ja<string>	任意	日本語ロケールメッセージ
- en<string>	任意	英語ロケールメッセージ
- zh_CN<string>	任意	中国語ロケールメッセージ
outputLog<boolean>	任意	ログ出力有無フラグ ユーザプログラム実行中にエラーが発生した際のログ出力有無を設定します。 このプロパティを省略した場合はログが出力されます。

コラム

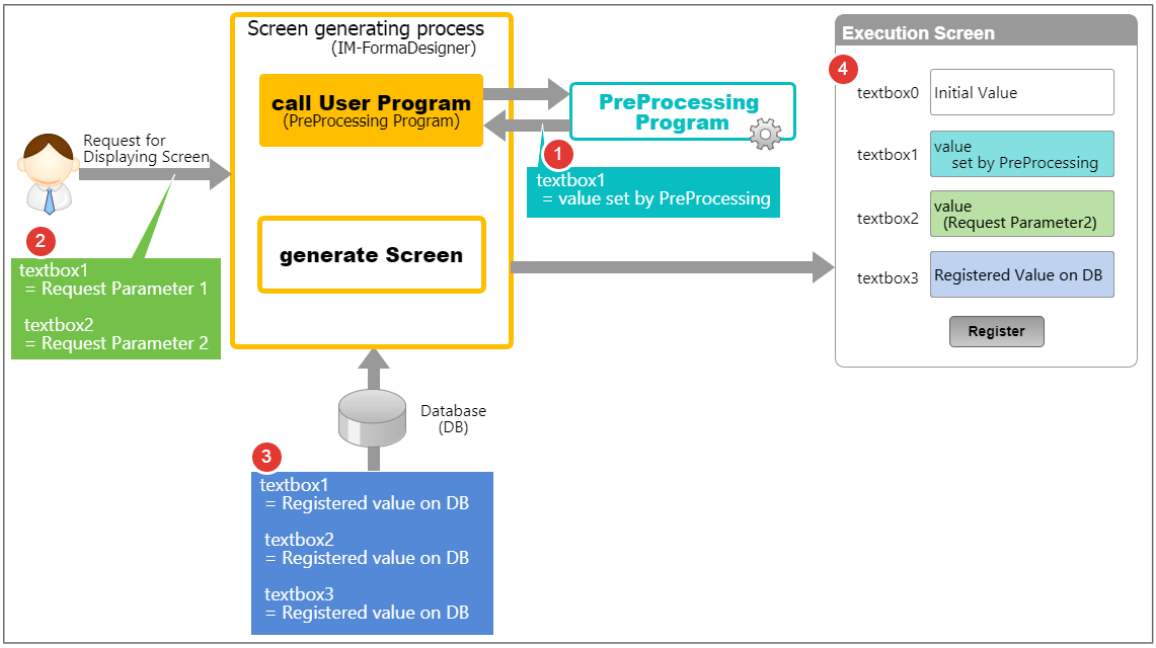
IM-LogicDesignerの「ロジックフロー定義編集」入出力設定では、JSON入力機能により以下のJSON文字列を取り込むことができます。

```
{
  "error": false,
  "errorMessage": "",
  "userErrorMessage": "",
  "localizedUserErrorMessages": {
    "ja": "",
    "en": "",
    "zh_CN": ""
  },
  "outputLog": true
}
```

前処理

IM-FormaDesigner for Accel Platform では、登録（申請）画面、編集（再申請）画面、詳細画面、承認画面など画面を表示するタイミング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。
前処理プログラムのしくみを利用することで入力系の画面アイテムに初期表示する値を指定することができます。

- 前処理が呼ばれる流れと値を設定するしくみ



1. 前処理プログラムの実行により、入力フィールドに表示する値（アイテム”textbox1”への値）が返却されます。
2. 画面表示のリクエストパラメータにより、アイテム”textbox1”、”textbox2”への値が設定されます。
   フォーム遷移を伴う場合、遷移元ページの入力値がリクエストパラメータにセットされます。
3. 更新・再申請など、すでにデータベースにデータが登録されている場合、データベースに登録されている値を返却します。
4. 表示された実行画面では、以下の優先順位に基づいて設定された値を初期表示します。
   - **設定の優先順位**
     1. 前処理プログラムによってセットされた値
     2. リクエストパラメータによってセットされた値
     3. データベースに登録されている値
     4. アイテムのプロパティ「フィールド初期値」にセットされている値
        ※「フィールド初期値」は登録（申請）時のみ有効です。

前処理プログラムの実装

JavaEE開発モデル

JavaEE開発モデルにおいて、前処理プログラムを実装する手順を示します。

実装規約

JavaEE開発モデルにおいて、前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。

- アプリケーション種別が「標準」の場合
  jp.co.intra_mart.foundation.forma.userprogram.StandardPreProcessExecutor 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

- 登録画面の表示時
  regist(final PreProcessParameter formaParam, final Map<String, Object> uppParam) メソッド
- 編集画面の表示時
  edit(final PreProcessParameter formaParam, final Map<String, Object> uppParam) メソッド
- 詳細画面の表示時
  refer(final PreProcessParameter formaParam, final Map<String, Object> uppParam) メソッド

- アプリケーション種別が「IM-Workflow」の場合
  jp.co.intra_mart.foundation.forma.userprogram.ImwPreProcessExecutor 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

  - 申請画面の表示時
    apply(final PreProcessParameter formaParam, final PreProcessWorkflowParameter workflowParam, final Map<String, Object> uppParam) メソッド
  - 再申請画面の表示時
    reapply(final PreProcessParameter formaParam, final PreProcessWorkflowParameter workflowParam, final Map<String, Object> uppParam) メソッド
  - 承認画面の表示時
    approve(final PreProcessParameter formaParam, final PreProcessWorkflowParameter workflowParam, final Map<String, Object> uppParam) メソッド
  - 詳細画面の表示時
    reference(final PreProcessParameter formaParam, final PreProcessWorkflowParameter workflowParam, final Map<String, Object> uppParam) メソッド

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。

jp.co.intra_mart.foundation.forma.userprogram.PreProcessParameter

IM-FormaDesigner for Accel Platformの基本情報を保持するクラスです。

メソッドの概要

メソッド	説明
String getLoginUserCd()	ログインユーザコードを返却します。
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（アプリケーション履歴番号）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
	
	IM-BIS の場合、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.BisApplicationType
	- **BIS_WKF**
	  ワークフロー
	- **BIS_BF**
	  BISフロー

メソッド	説明
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.ApplicationPageType ▪ **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）
String getProcessKey()	プロセスキーを返却します。
String getRecycleId()	リサイクルIDを返却します。（案件の再利用時のみ取得可能） 再利用元案件のアプリケーションIDを返却します。
String getRecycleDataId()	リサイクルデータIDを返却します。（案件の再利用時のみ取得可能） 再利用元案件のユーザデータIDを返却します。

jp.co.intra_mart.foundation.forma.userprogram.PreProcessWorkflowParameter

ワークフローに関するパラメータを保持するクラスです。
各画面で取得が可能な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」-「リクエストパラメータ」を参照してください。

メソッドの概要

メソッド	説明
String getImwApplyBaseDate()	申請基準日を返却します。
String getImwArriveType()	到達種別を返却します。
String getImwAuthUserCode()	権限者コードを返却します。
String getImwCallOriginalPagePath()	呼び出し元ページパスを返却します。
String getImwCallOriginalParams()	呼び出し元パラメータを返却します。
String getImwContentsId()	コンテンツIDを返却します。
String getImwContentsVersionId()	コンテンツバージョンIDを返却します。
String getImwFlowId()	フローIDを返却します。
String getImwFlowVersionId()	フローバージョンIDを返却します。
String getImwNodeId()	ノードIDを返却します。
String getImwPageType()	画面種別を返却します。
String getImwRouteId()	ルートIDを返却します。
String getImwRouteVersionId()	ルートバージョンIDを返却します。
String getImwSerialProcParams()	連続処理パラメータを返却します。
String getImwSystemMatterId()	システム案件IDを返却します。
String getImwUserCode()	処理者コードを返却します。
String getImwUserDataId()	ユーザデータIDを返却します。

uppParamマップ

画面呼び出し時にユーザプログラム用に渡されたリクエスト情報を保持するパラメータMap です。
パラメータ名の頭に「upp_」と付いている情報はユーザプログラム用のパラメータと判断され、受け渡しパラメータの対象と扱われます。

返却値

画面の初期表示値に任意の値を設定したい場合は返却値の更新データMapに値を設定して返却します。
更新データマップは Map<String, Object>形式で値を設定します。
画面表示時に同一フィールドに対して、前処理プログラムより与えられた更新データ、リクエストデータ、DBデータがあった場合に表示する値を決定する優先順位は以下の通りです。

前処理プログラムより与えられた更新データ ＞ リクエストデータ ＞ DBデータ

- Map<String, Object>のデータ構造
  - 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル、「ファイルアップロード」以外の画面アイテム
    キー ：フィールド識別ID
    値 ：表示する値
  - 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルのデータ
    キー ：テーブル識別ID
    値 ：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 です。
    各列のデータ型は列タイプに依存します。
  - 画面アイテム「ファイルアップロード」のデータ
    キー ：imfr_files
    値 ：jp.co.intra_mart.foundation.forma.userprogram.preprocess.AttachedFileクラスの配列
- 画面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	java.lang.Numberのサブクラス
日付またはタイムスタンプ	java.util.Date

jp.co.intra_mart.foundation.forma.userprogram.preprocess.AttachedFile

ファイルアップロード情報を保持するクラスです。

メソッドの概要

メソッド	説明
setFile(java.io.InputStream file)	ファイルを設定します。nullを指定した場合は、指定したアイテム識別IDの登録済みファイルを全てクリアします。
setFileId(java.lang.String fileId)	ユニークなファイルIDを設定します。
setFileName(java.lang.String fileName)	ファイル名を設定します。（必須）
setNotes(java.lang.String notes)	備考を設定します。
setUploadItemId(java.lang.String uploadItemId)	アイテム識別IDを設定します。（必須）

エラー処理

処理中にエラーが発生した場合は、 Exception を throw してください。
処理中にエラーが発生した場合に画面に任意のメッセージを表示したい場合は、FormaUserProcessException 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画面に遷移します。

jp.co.intra_mart.foundation.forma.exception.FormaUserProcessException

ユーザプログラム実行時例外クラスです。

主なメソッドの概要

メソッド	説明
void setUserMessageInfo(final String userMessageId, final String detailMessage)	画面に表示するメッセージID、詳細メッセージを指定します。（※詳細メッセージはメッセージIDに設定がある場合に適用されます）
void setOutputLog(final boolean isOutputLog)	ログ出力フラグを設定します。省略した場合はtrue（出力する）です。

トランザクションの制御

前処理プログラムはトランザクション内で実行されていません。
DB トランザクション制御を行う必要がある場合は、実行プログラム内においてトランザクションのコミット、ロールバック等を行ってください。

実装例

- アプリケーション種別が「標準」の場合

```java
public class SampleStandardPreProcess extends StandardPreProcessExecutor {

    /**
     * 登録画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> regist(final PreProcessParameter formaParam, final Map<String, Object> uppParam)
throws Exception {
        System.out.println("********** JAVAEE 開発：アプリケーション種別:標準用登録画面表示時前処理サンプルプログラム
**********");
        // ここに登録画面表示時に実行する処理を記述します。

        // ここに登録画面の各アイテムフィールドに任意の初期値を設定したい場合は、更新データMapを生成する処理を記述します。
        // 初期表示値の設定がない場合は、nullを返却してください。
        final Map<String, Object> updateData = new HashMap<String, Object>();
        // 文字列
        updateData.put("textbox1", "sample_setting_textbox1");
        // 数値
        updateData.put("number1", new Integer("10001"));
        // 日付
        updateData.put("calendar1", new Date());
        // チェックボックス
        updateData.put("checkbox1", "1,2");
        // テーブル識別IDが「table1」の明細テーブルに表示する値を設定する例
        final Map<String, Object> detaileTableData1 = new HashMap<String, Object>();
        detaileTableData1.put("table1_textbox1", "sample_setting_table_textbox1");
        detaileTableData1.put("table1_number1", new Integer("20001"));
        detaileTableData1.put("table1_calendar1", new Date());
        final Map<String, Object> detaileTableData2 = new HashMap<String, Object>();
        detaileTableData2.put("table1_textbox1", "sample_setting_table_textbox2");
        detaileTableData2.put("table1_number1", new Integer("20002"));
        detaileTableData2.put("table1_calendar1", new Date());
        final List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
        list.add(detaileTableData1);
        list.add(detaileTableData2);
        updateData.put("table1", list);

        // ファイルアップロード情報を設定
        final AttachedFile[] files = new AttachedFile[2];
        final AttachedFile file1 = new AttachedFile();
        final AttachedFile file2 = new AttachedFile();
        try {
            final InputStream in1 = new FileInputStream(new File("C://sample/sampleFile1.txt"));

            // 添付するファイルごとにFormaFileUpload型のモデルを作成します。
            file1.setFile(in1); // 添付ファイルの実体をjava.io.InputStream型で指定します。
            file1.setFileName("sampleFile1.txt"); // 添付ファイルのファイル名を指定します。
            file1.setUploadItemId("attach_fileupload_item1"); // アイテム識別IDを指定します。
            file1.setNotes("サンプルsampleのファイルです。");
            files[0] = file1;
            final InputStream in2 = new FileInputStream(new File("C://sample/sampleFile2.txt"));

            file2.setFile(in2);
            file2.setFileName("sampleFile2.txt");
            file2.setUploadItemId("attach_fileupload_item1");
            file2.setNotes("サンプルsample2のファイルです。");
            files[1] = file2;
            // ファイルアップロードアイテムデータをアプリケーションデータのパラメータにセットします。
            updateData.put("imfr_files", files);

        } catch (final IOException e) {
            // エラー処理
        }

        // エラー発生時に画面に表示するメッセージを指定する場合は、FormaUserProcessException をthrowします。
        if (XXXX) {
            final FormaUserProcessException error = new FormaUserProcessException(new XXXXException());
            error.setUserMessageInfo("画面に表示するメッセージID", "画面に表示する詳細メッセージ");
            throw error;
        }
        return updateData;
    }
```

```
    }

    /**
     * 編集画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> edit(final PreProcessParameter formaParam, final Map<String, Object> uppParam)
throws Exception {
        System.out.println("********** JAVAEE 開発：アプリケーション種別:編集画面表示時前処理サンプルプログラム
**********");
        // ここに編集画面表示時に実行する処理を記述します。
        return null;
    }

    /**
     * 参照画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> refer(final PreProcessParameter formaParam, final Map<String, Object> uppParam)
throws Exception {
        System.out.println("********** JAVAEE 開発：アプリケーション種別:参照画面表示時前処理サンプルプログラム
**********");
        // ここに参照画面表示時に実行する処理を記述します。
        return null;
    }
}
```

- アプリケーション種別が「IM-Workflow」の場合

```java
public class SampleWorkflowPreProcess extends ImwPreProcessExecutor {

    /**
     * 申請画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param workflowParam ワークフローパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> apply(final PreProcessParameter formaParam, final PreProcessWorkflowParameter
imwParameter, final Map<String, Object> uppParam) throws Exception {
        System.out.println("************** JAVAEE 開発：アプリケーション種別:IM-Workflow用申請画面表示時前処理サンプルプログ
ラム ****************");
        // ここに申請画面表示時に実行する処理を記述します。

        // ここに申請画面の各アイテムフィールドに任意の初期値を設定したい場合は、更新データMapを生成する処理を記述します。
        // 初期表示値の設定がない場合は、nullを返却してください。
        final Map<String, Object> updateData = new HashMap<String, Object>();
        // 文字列
        updateData.put("textbox1", "sample_setting_textbox1");
        // 数値
        updateData.put("number1", new Integer("10001"));
        // 日付
        updateData.put("calendar1", new Date());
        // チェックボックス
        updateData.put("checkbox1", "1,2");
        // テーブル識別IDが「table1」の明細テーブルに表示する値を設定する例
        final Map<String, Object> detaileTableData1 = new HashMap<String, Object>();
        detaileTableData1.put("table1_textbox1", "sample_setting_table_textbox1");
        detaileTableData1.put("table1_number1", new Integer("20001"));
        detaileTableData1.put("table1_calendar1", new Date());
        final Map<String, Object> detaileTableData2 = new HashMap<String, Object>();
        detaileTableData2.put("table1_textbox1", "sample_setting_table_textbox2");
        detaileTableData2.put("table1_number1", new Integer("20002"));
        detaileTableData2.put("table1_calendar1", new Date());
        final List<Map<String, Object>> list = new ArrayList<Map<String, Object>>();
        list.add(detaileTableData1);
        list.add(detaileTableData2);
        updateData.put("table1", list);

        // ファイルアップロード情報を設定
        final AttachedFile[] files = new AttachedFile[2];
        final AttachedFile file1 = new AttachedFile();
        final AttachedFile file2 = new AttachedFile();
        try {
            final InputStream in1 = new FileInputStream(new File("C://sample/sampleFile1.txt"));

            // 添付するファイルごとにFormaFileUpload型のモデルを作成します。
            file1.setFile(in1); // 添付ファイルの実体をjava.io.InputStream型で指定します。
            file1.setFileName("sampleFile1.txt"); // 添付ファイルのファイル名を指定します。
            file1.setUploadItemId("attach_fileupload_item1"); // アイテム識別IDを指定します。
            file1.setNotes("サンプルsampleのファイルです。");
            files[0] = file1;
            final InputStream in2 = new FileInputStream(new File("C://sample/sampleFile2.txt"));

            file2.setFile(in2);
            file2.setFileName("sampleFile2.txt");
            file2.setUploadItemId("attach_fileupload_item1");
            file2.setNotes("サンプルsample2のファイルです。");
            files[1] = file2;
            // ファイルアップロードアイテムデータをアプリケーションデータのパラメータにセットします。
            updateData.put("imfr_files", files);

        } catch (final IOException e) {
            // エラー処理
        }

        // エラー発生時に画面に表示するメッセージを指定する場合は、FormaUserProcessException をthrowします。
        if (XXXX) {
            final FormaUserProcessException error = new FormaUserProcessException(new XXXXException());
            error.setUserMessageInfo("画面に表示するメッセージID", "画面に表示する詳細メッセージ");
            throw error;
        }

```

```
        return updateData;
    }

    /**
     * 承認画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param workflowParam ワークフローパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> approve(final PreProcessParameter formaParam, final PreProcessWorkflowParameter
imwParameter, final Map<String, Object> uppParam) throws Exception {
        System.out.println("********** JAVAEE 開発：アプリケーション種別:IM-Workflow用承認画面表示時前処理サンプルプログ
ラム **********");
        // ここに承認画面表示時に実行する処理を記述します。
        return null;

    }

    /**
     * 再申請画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param workflowParam ワークフローパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> reapply(final PreProcessParameter formaParam, final PreProcessWorkflowParameter
imwParameter, final Map<String, Object> uppParam) throws Exception {
        System.out.println("********** JAVAEE 開発：アプリケーション種別:IM-Workflow用再申請画面表示時前処理サンプルプロ
グラム **********");
        // ここに再申請画面表示時に実行する処理を記述します。
        return null;
    }

    /**
     * 参照画面表示時に実行されます。<BR>
     * @param formaParam フォルマパラメータ
     * @param workflowParam ワークフローパラメータ
     * @param uppParam ユーザプログラムパラメータ
     * @return 更新データ
     * @throws Exception 例外が発生
     */
    @Override
    public Map<String, Object> reference(final PreProcessParameter formaParam, final PreProcessWorkflowParameter
imwParameter, final Map<String, Object> uppParam) throws Exception {
        System.out.println("********** JAVAEE 開発：アプリケーション種別:IM-Workflow用参照画面表示時前処理サンプルプログ
ラム **********");
        // ここに参照画面表示時に実行する処理を記述します。
        return null;
    }

}
```

スクリプト開発モデル

スクリプト開発モデルにおいて、前処理プログラムを実装する手順を示します。

実装規約

スクリプト開発モデルにおいて、前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。

- アプリケーション種別が「標準」の場合
  画面を表示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
  - 登録画面の表示時
    regist(formaParam, uppParam) メソッド
  - 編集画面の表示時
    edit(formaParam, uppParam) メソッド

  - 詳細画面の表示時
    refer(formaParam, uppParam) メソッド
- アプリケーション種別が「IM-Workflow」の場合
  画面を表示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
  - 申請画面の表示時
    apply(formaParam, imwParam, uppParam) メソッド
  - 再申請画面の表示時
    reapply(formaParam, imwParam, uppParam) メソッド
  - 承認画面の表示時
    approve(formaParam, imwParam, uppParam) メソッド
  - 詳細画面の表示時
    reference(formaParam, imwParam, uppParam) メソッド

パラメータ

formaParamオブジェクト

IM-FormaDesigner の画面の基本情報を保持します。

項目	説明
loginUserCd(String)	ログインユーザコード
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（アプリケーション履歴番号）
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 IM-BIS の場合、アプリケーション種別は BisApplication オブジェクトに定義されています。 ▪ **BIS_APPLICATION_TYPE_WORKFLOW** ワークフロー ▪ **BIS_APPLICATION_TYPE_BUSINESSFLOW** BISフロー
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）
processKey(String)	プロセスキー
recycleId(String)	リサイクルIDを返却します。（案件の再利用時のみ取得可能） 再利用元案件のアプリケーションIDを返却します。
recycleDataId(String)	リサイクルデータIDを返却します。（案件の再利用時のみ取得可能） 再利用元案件のユーザデータIDを返却します。

imwParamオブジェクト

ワークフローに関するパラメータを保持します。
各画面で取得が可能な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 」を参照してください。

プロパティの概要

項目	説明
imwApplyBaseDate(String)	申請基準日
imwArriveType(String)	到達種別
imwAuthUserCode(String)	権限者コード
imwCallOriginalPagePath(String)	呼び出し元ページパス
imwCallOriginalParams(String)	呼び出し元パラメータ
imwContentsId(String)	コンテンツID
imwContentsVersionId(String)	コンテンツバージョンID
imwFlowId(String)	フローID
imwFlowVersionId(String)	フローバージョンID
imwNodeId(String)	ノードID
imwPageType(String)	画面種別
imwRouteId(String)	ルートID
imwRouteVersionId(String)	ルートバージョンID
imwSerialProcParams(String)	連続処理パラメータ
imwSystemMatterId(String)	システム案件ID
imwUserCode(String)	処理者コード
imwUserDataId(String)	ユーザデータID

uppParamオブジェクト

画面呼び出し時にユーザプログラム用に渡されたリクエスト情報を保持します。
パラメータ名の頭に「upp_」と付いている情報がユーザプログラム用のパラメータと判断され受け渡しパラメータの対象として扱われます。

返却値とエラー処理

```
処理結果オブジェクト
 ├ data：更新データオブジェクト
 ├ files：更新ファイルアップロードデータオブジェクト
 ├ 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。）
 ├ userErrorMessageId：画面に表示するメッセージのID
 ├ userErrorMessage：画面に表示する詳細メッセージ（※userErrorMessageIdに設定がある場合に適用されます）
 └ outputLog：ログ出力有無フラグ　ユーザプログラム実行中にエラーが発生した際のログ出力有無を設定します。　省略した場合はtrue（出力する）です。
```

- 返却値
  画面の入力フィールドに表示する初期表示値を任意の値に設定したい場合は更新データオブジェクトに値を設定します。
  画面表示時に同一フィールドに対して、前処理プログラムより与えられた更新データ、リクエストデータ、DBデータがあった場合に表示する値を決定する優先順位は以下の通りです。

  前処理プログラムより与えられた更新データ > リクエストデータ > DBデータ

- 更新データオブジェクト
  - 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル、「ファイルアップロード」以外の画面アイテム | | プロパティ：フィールド識別ID | 値 ：表示する値
  - 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルのデータ
    プロパティ：テーブル識別ID

    値 ：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。

- 画面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	String
数値	Number
日付またはタイムスタンプ	Date

- 更新ファイルアップロードデータオブジェクト
  - ファイルアップロードオブジェクトの配列

    プロパティの概要

項目	説明
filePath(String)	パブリックストレージのファイルパス ファイルパスに空文字を指定した場合は、指定したアイテム識別IDの登録済みファイルを全てクリアします。 実体がないファイルパスを指定した場合は、警告ログを出力して処理をスキップします。
uploadItemId(String)	アイテム識別ID
notes(String)	備考

  - エラー発生時
    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画面に遷移します。
    画面に表示するメッセージIDの指定がなかった場合は、エラー画面にはデフォルトのエラーメッセージが表示されます。

トランザクションの制御

前処理プログラムはトランザクション内で実行されていません。
DB トランザクション制御を行う必要がある場合は、実行プログラム内においてトランザクションのコミット、ロールバック等を行ってください。

実装例

- アプリケーション種別が「標準」の場合

```
// 登録
function regist(formaParam, uppParam) {
    Debug.print('********** スクリプト開発：アプリケーション種別:標準用登録画面表示時前処理サンプルプログラム
**********');
    // ここに登録画面表示時に実行する処理を記述します。

    // ここに登録画面の各アイテムフィールドに任意の初期値を設定したい場合は、更新データオブジェクトを生成する処理を記述します。
    // 初期表示値の設定がない場合は空を返却してください。

    var obj = {
      'textbox1': 'sample_setting_textbox1',
      'number1': 10001,
      'date1' : new Date(),
      'checkbox1' :'1,2',
      'table1' : [
          {
           'table1_textbox1': 'sample_setting_table_textbox1',
           'table1_number1': 20001,
           'table1_date1' : new Date()
          },
          {
           'table1_textbox1': 'sample_setting_table_textbox2',
           'table1_number1': 20002,
           'table1_date1' : new Date()
          }
        ]
       };

    // 添付するファイルごとにオブジェクトを作成します。
    var file1 = {};
    file1.filePath = 'sample/sampleFile1.txt'; // パブリックストレージのファイルパスを指定します。
    file1.uploadItemId = 'attach_fileupload_item1'; // アイテム識別IDを指定します。
    file1.notes = 'サンプルのファイル1です。';

    var file2 = {};
    file2.filePath = 'sample/sampleFile2.txt';
    file2.uploadItemId = 'attach_fileupload_item1';
    file2.notes = 'サンプルのファイル2です。';

    return {'error' : false, data : obj,  'files' : [file1, file2]};
}

// 編集
function edit(formaParam, uppParam) {
    Debug.print('********** スクリプト 開発：アプリケーション種別:編集画面表示時前処理サンプルプログラム **********');
    Debug.print('---formaParam-----');
    Debug.console(formaParam);
    Debug.print('---uppParam-----');
    Debug.console(uppParam);

    var obj = {
      'textbox2': 'u_data_ee',
      'number2': 22222,
      'date2' : new Date()
    };
    return {'error' : false, data : obj};
}

// 参照
function refer(formaParam, uppParam) {
    Debug.print('********** スクリプト 開発：アプリケーション種別:参照画面表示時前処理サンプルプログラム **********');
    Debug.print('---formaParam-----');
    Debug.console(formaParam);
    Debug.print('---uppParam-----');
    Debug.console(uppParam);

    var obj = {
      'textbox3': 'u_data_eee',
      'number3': 33333,
      'date3' : new Date()
    };
    return {'error' : false, data : obj};
}
```

- アプリケーション種別が「IM-Workflow」の場合

```
// 申請
function apply(formaParam, imwParam, uppParam)  {
    Debug.print('********** スクリプト 開発：アプリケーション種別:IM-Workflow用申請画面表示時前処理サンプルプログラム
**********');
    Debug.print('---formaParam-----');
    Debug.console(formaParam);
    Debug.print('---imwParam-----');
    Debug.console(imwParam);
    Debug.print('---uppParam-----');
    Debug.console(uppParam);

    var obj = {
      'textbox1': 'u_data_e',
      'number1': 11111,
      'date1' : new Date()
    };

    // 添付するファイルごとにオブジェクトを作成します。
    var file1 = {};
    file1.filePath = 'sample/sampleFile1.txt'; // パブリックストレージのファイルパスを指定します。
    file1.uploadItemId = 'attach_fileupload_item1'; // アイテム識別IDを指定します。
    file1.notes = 'サンプルのファイル1です。';

    var file2 = {};
    file2.filePath = 'sample/sampleFile2.txt';
    file2.uploadItemId = 'attach_fileupload_item1';
    file2.notes = 'サンプルのファイル2です。';

    return {'error' : false, data : obj,  'files' : [file1, file2]};
}

// 再申請
function reapply(formaParam, imwParam, uppParam) {
    Debug.print('********** スクリプト 開発：アプリケーション種別:IM-Workflow用再申請画面表示時前処理サンプルプログラム
**********');
    Debug.print('---formaParam-----');
    Debug.console(formaParam);
    Debug.print('---imwParam-----');
    Debug.console(imwParam);
    Debug.print('---uppParam-----');
    Debug.console(uppParam);

    var obj = {
      'textbox1': 'u_data_e_',
      'number1': 111110,
      'date1' : new Date()
    };
    return {'error' : false, data : obj};
}

// 承認
function approve(formaParam, imwParam, uppParam) {
    Debug.print('********** スクリプト 開発：アプリケーション種別:IM-Workflow用承認画面表示時前処理サンプルプログラム
**********');
    Debug.print('---formaParam-----');
    Debug.console(formaParam);
    Debug.print('---imwParam-----');
    Debug.console(imwParam);
    Debug.print('---uppParam-----');
    Debug.console(uppParam);

    var obj = {
      'textbox2': 'u_data_ee',
      'number2': 22222,
      'date2' : new Date()
    };
    return {'error' : false, data : obj};
}


// 参照
function reference(formaParam, imwParam, uppParam) {
    Debug.print('********** スクリプト 開発：アプリケーション種別:IM-Workflow用参照画面表示時前処理サンプルプログラム
**********');
    Debug.print('---formaParam-----');
    Debug.console(formaParam);
```

```
    Debug.print('---imwParam-----');
    Debug.console(imwParam);
    Debug.print('---uppParam-----');
    Debug.console(uppParam);

    var obj = {
      'textbox3': 'u_data_eee',
      'number3': 33333,
      'date3' : new Date()
    };
    return {'error' : false, data : obj};
}
```

ロジックフロー

ロジックフローにおいて、前処理プログラムを実装する手順を示します。

IM-LogicDesignerの操作方法につきましては「IM-LogicDesigner ユーザ操作ガイド」を参照してください。

入力値

ロジックフローの入力値に渡される入力パラメータ情報です。

```
executeProcessType <string>
preProcessParameter <object>
 ├ loginUserCd <string>
 ├ applicationId <string>
 ├ applicationNo <long>
 ├ insertId <string>
 ├ applicationType <string>
 ├ appPageType <string>
 ├ processKey <string>
 ├ recycleId <string>
 └ recycleDataId <string>
uppParam <object>
preProcessWorkflowParameter <object>
 ├ imwApplyBaseDate <string>
 ├ imwArriveType <string>
 ├ imwAuthUserCode <string>
 ├ imwCallOriginalPagePath <string>
 ├ imwCallOriginalParams <string>
 ├ imwContentsId <string>
 ├ imwContentsVersionId <string>
 ├ imwFlowId <string>
 ├ imwFlowVersionId <string>
 ├ imwNodeId <string>
 ├ imwPageType <string>
 ├ imwRouteId <string>
 ├ imwRouteVersionId <string>
 ├ imwSerialProcParams <string>
 ├ imwSystemMatterId <string>
 ├ imwUserCode <string>
 └ imwUserDataId <string>
imbpmExecutionInfo <object>
 ├ id <string>
 ├ processInstanceId <string>
 ├ businessKey <string>
 ├ processDefinitionId <string>
 ├ superExecutionId <string>
 ├ currentActivityId <string>
 ├ currentActivityName <string>
 ├ variablesLocal <object>
 └ variables <object>
```

入力値	説明
executeProcessType<string>	*executeProcessType* を参照
preProcessParameter<object>	*preProcessParameterオブジェクト* を参照
uppParam<object>	*uppParamオブジェクト* を参照
preProcessWorkflowParameter<object>	*preProcessWorkflowParameterオブジェクト* を参照

入力値	説明
imbpmExecutionInfo<object>	*imbpmExecutionInfoオブジェクト* を参照

コラム

IM-LogicDesignerの「ロジックフロー定義編集」入出力設定では、JSON入力機能により以下のJSON文字列を取り込むことができます。

```
{
 "executeProcessType": "",
 "preProcessParameter": {
  "loginUserCd": "",
  "applicationId": "",
  "applicationNo": 0,
  "insertId": "",
  "applicationType": "",
  "appPageType": "",
  "processKey": "",
  "recycleId": "",
  "recycleDataId": ""
 },
 "uppParam": {},
 "preProcessWorkflowParameter": {
  "imwApplyBaseDate": "",
  "imwArriveType": "",
  "imwAuthUserCode": "",
  "imwCallOriginalPagePath": "",
  "imwCallOriginalParams": "",
  "imwContentsId": "",
  "imwContentsVersionId": "",
  "imwFlowId": "",
  "imwFlowVersionId": "",
  "imwNodeId": "",
  "imwPageType": "",
  "imwRouteId": "",
  "imwRouteVersionId": "",
  "imwSerialProcParams": "",
  "imwSystemMatterId": "",
  "imwUserCode": "",
  "imwUserDataId": ""
 },
 "imbpmExecutionInfo": {
  "id": "",
  "processInstanceId": "",
  "businessKey": "",
  "processDefinitionId": "",
  "superExecutionId": "",
  "currentActivityId": "",
  "currentActivityName": "",
  "variablesLocal": {},
  "variables": {}
 }
}
```

executeProcessType

アプリケーション実行種別を保持します。
画面を表示するタイミングに合わせて以下の値が取得可能です。

- アプリケーション種別が「標準」の場合
  - **regist**
    登録画面の表示時
  - **edit**
    編集画面の表示時
  - **refer**
    詳細画面の表示時
- アプリケーション種別が「IM-Workflow」の場合
  - **apply**

申請画面の表示時

- **reapply**
  再申請画面の表示時
- **approve**
  承認画面の表示時
- **reference**
  詳細画面の表示時

preProcessParameterオブジェクト

IM-FormaDesigner の画面の基本情報を保持します。

項目	説明
loginUserCd<string>	ログインユーザコード
applicationId<string>	アプリケーションID
applicationNo<long>	アプリケーションバージョンNO（アプリケーション履歴番号）
insertId<string>	データ登録ID
applicationType<string>	アプリケーション種別 - **std** 標準 - **imw** IM-Workflow IM-BIS の場合 - **bis_wkf** ワークフロー - **bis_bf** BISフロー
appPageType<string>	アプリケーションページ種別 - **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）
processKey<string>	プロセスキー
recycleId<string>	リサイクルIDを返却します。（案件の再利用時のみ取得可能） 再利用元案件のアプリケーションIDを返却します。
recycleDataId<string>	リサイクルデータIDを返却します。（案件の再利用時のみ取得可能） 再利用元案件のユーザデータIDを返却します。

uppParamオブジェクト

画面呼び出し時にユーザプログラム用に渡されたリクエスト情報を保持します。
パラメータ名の頭に「upp_」と付いている情報がユーザプログラム用のパラメータと判断され受け渡しパラメータの対象として扱われます。

preProcessWorkflowParameterオブジェクト

ワークフローに関するパラメータを保持します。
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が利用できます。

各画面で取得が可能な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」-「リクエストパラメータ」を参照してください。

プロパティの概要

入力値	説明
imwApplyBaseDate<string>	申請基準日
imwArriveType<string>	到達種別
imwAuthUserCode<string>	権限者コード
imwCallOriginalPagePath<string>	呼び出し元ページパス
imwCallOriginalParams<string>	呼び出し元パラメータ
imwContentsId<string>	コンテンツID
imwContentsVersionId<string>	コンテンツバージョンID
imwFlowId<string>	フローID
imwFlowVersionId<string>	フローバージョンID
imwNodeId<string>	ノードID
imwPageType<string>	画面種別
imwRouteId<string>	ルートID
imwRouteVersionId<string>	ルートバージョンID
imwSerialProcParams<string>	連続処理パラメータ
imwSystemMatterId<string>	システム案件ID
imwUserCode<string>	処理者コード
imwUserDataId<string>	ユーザデータID

imbpmExecutionInfoオブジェクト

案件の連携元タスクのエグゼキューション情報を保持します。
IM-BPM for Accel Platformの申請タスクまたは起票タスクからIM-Workflowが実行された場合利用できます。

詳細については、「IM-BPM プロセスデザイナ 操作ガイド」 -「申請タスク」または 「起票タスク」を参照してください。

入力値	説明
id<string>	エグゼキューションID
processInstanceId<string>	プロセスインスタンスID
businessKey<string>	業務キー
processDefinitionId<string>	プロセス定義ID
superExecutionId<string>	親プロセスのエグゼキューションID
currentActivityId<string>	タスクのアクティビティID
currentActivityName<string>	タスクのアクティビティ名
variablesLocal<object>	変数（スコープがエグゼキューションであるもの） ・ 変数名をキー、変数値を値として持つ object です。
variables<object>	変数（スコープがプロセスインスタンスであるもの） ・ 変数名をキー、変数値を値として持つ object です。

出力値

ロジックフローの出力値に設定する情報です。

```
data <object>
files <object[]>
 ├ file <storage>
 ├ uploadItemId <string>
 └ notes <string>
error <boolean>
errorMessage <string>
userErrorTitleMessage <string>
localizedUserErrorTitleMessages <object>
 ├ ja <string>
 ├ en <string>
 └ zh_CN <string>
userErrorMessage <string>
localizedUserErrorMessages <object>
 ├ ja <string>
 ├ en <string>
 └ zh_CN <string>
outputLog <boolean>
```

説明

画面アイテムの更新

- 画面の入力フィールドに表示する初期表示値を任意の値に設定したい場合 `data<object>` に値を設定します。
- 画面表示時に同一フィールドに対して、前処理プログラムより与えられた更新データ、リクエストデータ、DBデータがあった場合に表示する値を決定する優先順位は以下の通りです。
  - **前処理プログラムより与えられた更新データ > リクエストデータ > DBデータ**
- 更新対象の画面アイテムは `data<object>` に以下のように設定します。
  - 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブル、「ファイルアップロード」以外の画面アイテム
    キー名：フィールド識別ID
    値 ：表示する値
  - 画面アイテム「明細テーブル」、「グリッドテーブル」、「スプレッドシート」のテーブルのデータ
    キー名：テーブル識別ID
    値 ：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 `<object[]>` です。
    各列のデータ型は列タイプに依存します。

    例）

    ```
    data <object>
     ├ textbox1 <string>
     ├ number1 <integer>
     ├ calendar1 <date>
     └ tb1 <object[]>
       ├ [0] // テーブル1行目
       │  ├ tb1_textbox1 <string>
       │  └ tb1_number1 <integer>
       ├ [1] // テーブル2行目
       │  ├ tb1_textbox1 <string>
       │  └ tb1_number1 <integer>
       └ [2] // テーブル3行目
          ├ tb1_textbox1 <string>
          └ tb1_number1 <integer>
    ```

- 画面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	`<string>`
数値	`<integer>` 、 `<long>` 、 `<double>` を含めたキャスト可能な数値型
日付またはタイムスタンプ	`<date>`

- 更新ファイルアップロードデータオブジェクト配列
  - `files<object[]>`

    プロパティの概要

項目	説明
file<storage>	ファイルの取得対象となるストレージ。 何も設定しない場合は、設定したアイテム識別IDの登録済みファイルを全てクリアします。 実体がないファイルを設定した場合は、警告ログを出力して処理をスキップします。
uploadItemId<string>	アイテム識別ID
notes<string>	備考

処理エラー時の動作

- 処理中にエラーが発生した場合は error<boolean> に true 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画面に遷移します。
  - 画面に表示するエラーメッセージの設定がなかった場合は、エラー画面にはデフォルトのエラーメッセージが表示されます。
  - 例外ログの出力有無は outputLog<boolean> で設定できます。

プロパティ

出力値	必須/任意	説明
data<object>	任意	更新データオブジェクト
files<object[]>	任意	更新ファイルアップロードデータオブジェクト配列
- file<storage>	任意	ファイルの取得対象となるストレージを設定します。 何も設定しない場合は、設定したアイテム識別IDの登録済みファイルを全てクリアします。 実体がないファイルを設定した場合は、警告ログを出力して処理をスキップします。
- uploadItemId<string>	任意	アイテム識別ID
- notes<string>	任意	備考
error<boolean>	必須	エラーフラグ true を設定した場合、当処理をエラー終了します。 処理が正常に終了した場合は false を設定してください。
errorMessage<string>	任意	設定したメッセージを例外ログとして出力します。
userErrorTitleMessage<string>	任意	設定したメッセージをエラー画面の一行目に表示します。 ロケールごとにメッセージを用意する必要がなければこのプロパティを利用してください。 このプロパティを省略した場合、エラー画面にはデフォルトのエラーメッセージが表示されます。
localizedUserErrorTitleMessages<object>	任意	設定したメッセージをロケールに従ってエラー画面の一行目に表示します。 アカウントコンテキストから解決されたロケールに一致するメッセージを使用します。 userErrorTitleMessage が設定されている場合はこのプロパティは使用されません。
- ja<string>	任意	日本語ロケールメッセージ
- en<string>	任意	英語ロケールメッセージ
- zh_CN<string>	任意	中国語ロケールメッセージ
userErrorMessage<string>	任意	設定したメッセージをエラー画面の二行目に表示します。 ロケールごとにメッセージを用意する必要がなければこのプロパティを利用してください。 userErrorTitleMessage が設定されている場合に適用されます。
localizedUserErrorMessages<object>	任意	設定したメッセージをロケールに従ってエラー画面の二行目に表示します。 アカウントコンテキストから解決されたロケールに一致するメッセージを使用します。 userErrorMessage が設定されている場合はこのプロパティは使用されません。
- ja<string>	任意	日本語ロケールメッセージ

出力値	必須/任意	説明
- en<string>	任意	英語ロケールメッセージ
- zh_CN<string>	任意	中国語ロケールメッセージ
outputLog<boolean>	任意	ログ出力有無フラグ ユーザプログラム実行中にエラーが発生した際のログ出力有無を設定します。 このプロパティを省略した場合はログが出力されます。

コラム

IM-LogicDesignerの「ロジックフロー定義編集」入出力設定では、JSON入力機能により以下のJSON文字列を取り込むことができます。
この時、「null値の型」として storage を設定してください。

```
{
 "data": {},
 "files": [
  {
   "file": null,
   "uploadItemId": "",
   "notes": ""
  }
 ],
 "error": false,
 "errorMessage": "",
 "userErrorTitleMessage": "",
 "localizedUserErrorTitleMessages": {
  "ja": "",
  "en": "",
  "zh_CN": ""
 },
 "userErrorMessage": "",
 "localizedUserErrorMessages": {
  "ja": "",
  "en": "",
  "zh_CN": ""
 },
 "outputLog": true
}
```

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注意してください。

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

コンストラクタ

- コンストラクタ①
  - ApplicationDataManager(String loginUserCd)
    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

- コンストラクタ②
  - ApplicationDataManager(String loginUserCd, 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できます。

パラメータ	
loginUserCd	ログインユーザコード
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

主なメソッド

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

パラメータ	
applicationId	アプリケーションID
insertId	登録データ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

返却値
登録情報マップ　※詳細は後述の*登録情報マップ*を参照してください。

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

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（アプリケーション履歴番号）
insertId	登録データID
itemData	登録情報マップ ※詳細は後述の*登録情報マップ*を参照してください。

- 返却値

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。

> **コラム**
>
> 登録データIDはアプリケーションのデータを一意に識別するためのキーです。
> jp.co.intra_mart.foundation.service.client.information.Identifier#get() を利用して採番してください。

データの更新メソッド

- void updateItemData(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

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（アプリケーション履歴番号）
insertId	登録データID
itemData	登録情報マップ ※詳細は後述の*登録情報マップ*を参照してください。

  - 返却値

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。

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

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

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。

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>
- 画面アイテム「明細テーブル」、「グリッドテーブル」以外の画面アイテム ※「スプレッドシート」のテーブル定義のデータを除く
  キー ：フィールド識別ID
  値： 入力値
- 画面アイテム「明細テーブル」、「グリッドテーブル」のデータ、「スプレッドシート」のテーブル定義のデータ
  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 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。
  ※画面アイテム「ファイルアップロード」の情報は登録データに含まれません。
- 画面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	*＜データの登録・更新時＞と画面アイテム「スプレッドシート」の場合* java.util.Date *画面アイテム「スプレッドシート」以外の＜データ取得時＞* java.sql.Date
タイムスタンプ	java.sql.Timestamp

> **コラム**
>
> API利用時に、メソッドの引数として設定した登録情報マップは、後処理プログラムの引数となるsendParamマップへは連携されません。

スクリプト開発モデル

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オブジェクトが用意されています。

- FRApplicationDataManager

コンストラクタ

- コンストラクタ①
  - FRApplicationDataManager(String loginUserCd)
    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

パラメータ
loginUserCd　ログインユーザコード

- コンストラクタ②
  - FRApplicationDataManager(String loginUserCd, 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できます。
    - パラメータ

パラメータ	
loginUserCd	ログインユーザコード
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

主なメソッド

データの取得メソッド

- ScriptableObject 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

パラメータ	
applicationId	アプリケーションID
insertId	登録データ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
    下記の情報を持つオブジェクトを返します。

返却値	
data	登録情報オブジェクト ※詳細は後述の*ItemDataObject 登録情報オブジェクト*を参照してください。
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を返します。
errorMessage	エラーメッセージ

データの登録メソッド

- ResultObject 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

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（アプリケーション履歴番号）
insertId	登録データID
itemDataObject	登録情報オブジェクト ※詳細は後述の*ItemDataObject 登録情報オブジェクト*を参照してください。

  - 返却値

返却値
処理結果オブジェクト　※詳細は後述の*ResultObject 処理結果オブジェクト*を参照してください。

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。

> **コラム**
>
> 登録データIDはアプリケーションのデータを一意に識別するためのキーです。
> Identifierオブジェクトのgetメソッド を利用して採番してください。

データの更新メソッド

- ResultObject updateItemData(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

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（アプリケーション履歴番号）
insertId	登録データID
ItemDataObject	登録情報オブジェクト ※詳細は後述の*ItemDataObject 登録情報オブジェクト*を参照してください。

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。

データの削除メソッド

- ResultObject 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

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

返却値
処理結果オブジェクト　※詳細は後述の*ResultObject 処理結果オブジェクト*を参照してください。

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。

ItemDataObject 登録情報オブジェクト

- 画面アイテム「明細テーブル」、「グリッドテーブル」以外の画面アイテム ※「スプレッドシート」のテーブル定義のデータを除く
  プロパティ：フィールド識別ID
- 画面アイテム「明細テーブル」、「グリッドテーブル」のデータ、「スプレッドシート」のテーブル定義のデータ
  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。
  ※画面アイテム「ファイルアップロード」の情報には登録データには含まれません。
- 画面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	String
数値	Number
日付またはタイムスタンプ	Date

コラム

API利用時に、メソッドの引数として設定した登録情報オブジェクトは、後処理プログラムの引数となるsendParamマップへは連携されません。

ResultObject 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を返します。
- exclusionError：排他エラーフラグ
  更新、削除時に排他エラーが発生した場合はtrue、以外はfalseを返します。
- errorMessage: エラーメッセージ

PDF出力処理

IM-FormaDesigner for Accel Platform では、参照画面をPDFで出力する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。
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。

後処理プログラムでの実装手順

> **コラム**
>
> 当ページでは「wkhtmltopdf を利用したPDF出力機能」について説明します。

> **注意**
>
> wkhtmltopdf は2023年1月にアーカイブされました。
> そのため、wkhtmltopdf の運用は非推奨となりました。
>
> wkhtmltopdf の代替方法は以下のURLを参照してください。
> https://product.intra-mart.support/hc/ja/articles/36748979822873

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

PDF出力するAPIは、以下の箇所で利用できます。

- 後処理プログラムデータの更新時、データの削除時

> **コラム**
>
> PDF出力処理は、API を実行するログインユーザのセッションを利用して行われます。
> そのため、参照画面にログインユーザのセッション情報（ユーザ名等）を表示する画面アイテムが配置されている場合、実行したユーザのセッション情報が出力されます。

JavaEE開発モデル

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。

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を推奨します。

実装例1：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;


    // 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();
        }
    }
```

実装例2：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;

    // 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();
        }
    }
```

- PDFファイルの出力方向を標準の縦方向から横方向に変更する場合、オプション指定で以下の内容を記述してください。

```
options.add("--orientation");
options.add("Landscape");
```

> **コラム**
>
> オプションに設定値がある場合は、上記のようにoptionsに分けて設定してください。

> **コラム**
>
> ロードバランシング環境にて、改善モジュールを利用している場合は、APIの引数としてオプションを指定する必要があります。
> オプションの設定値については、以下を参照ください。
>
> - FAQ詳細ページ

スクリプト開発モデル

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。

- 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を推奨します。

実装例1：オプション指定なし

```
1  // パラメータ
2  var applicationId = "XX"; // アプリケーションID
3  var insertId      = "XX"; // ユーザデータID
4
5  // PDF出力処理
6  var pdfConverter = new FRApplicationPDFConverter();
7  var result = pdfConverter.createPDF(applicationId, insertId);
8  if (result.error) {
9    // TODO エラー処理
10   Debug.console(result);
11 } else {
12   // PDF ファイル
13   var pdfFile = result.data;
14
15   : (ファイル操作処理)
16
17   // 出力されたPDF ファイルを削除
18   pdfFile.remove();
19 }
```

実装例2：オプション指定あり

```
// 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, options);
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();
}
```

- PDFファイルの出力方向を標準の縦方向から横方向に変更する場合、オプション指定で以下の内容を記述してください。

```
options.push("--orientation");
options.push("Landscape");
```

> **コラム**
>
> オプションに設定値がある場合は、上記のように分けてoptionsに分けて設定してください。

> **コラム**
>
> ロードバランシング環境にて、改善モジュールを利用している場合は、APIの引数としてオプションを指定する必要があります。
> オプションの設定値については、以下を参照ください。
>
> - FAQ詳細ページ

ジョブスケジューラ・非同期処理機能での実装手順

> **コラム**
>
> 当ページでは「wkhtmltopdf を利用したPDF出力機能」について説明します。

> **注意**
>
> wkhtmltopdf は2023年1月にアーカイブされました。
> そのため、wkhtmltopdf の運用は非推奨となりました。
>
> wkhtmltopdf の代替方法は以下のURLを参照してください。
> https://product.intra-mart.support/hc/ja/articles/36748979822873

ジョブスケジューラ・非同期処理機能にて、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
- 対象の参照画面を表示可能なユーザのユーザコード

PDF出力するAPIは、以下の機能で利用できます。
ジョブスケジューラ・非同期処理機能の実装方法については、それぞれのドキュメントを参照してください。

- ジョブネット （ ジョブスケジューラ仕様書 ）
- 非同期処理 （ 非同期仕様書 ）

コラム

参照画面をPDF出力する API では、処理内部でショートカットURLを作成します。
そのため、対象の参照画面を表示可能なユーザのユーザコードを指定する必要があります。

コラム

ワークフローの詳細画面をPDF出力する場合は、対象の案件が完了している必要があります。
そのため、IM-Workflow の到達処理や案件終了処理では、直接利用することはできません。
案件終了処理から非同期処理としてや、ジョブスケジューラにて利用してください。
ワークフローの案件の状態は、「 UserMatterStatusオブジェクト 」で確認することできます。

コラム

複数フォーム画面をPDF出力する場合、「wkhtmltopdf」のバージョンによっては、各画面毎に横幅を基準に出力サイズを自動調整します。
そのため、各画面毎に毎に異なった拡大率で出力される場合があります。
変更する場合は、「wkhtmltopdf」のオプションを指定してください。

注意

Office 365 連携 でPDFを出力する場合、javascript-delayオプションが必要となるため、wkhtmltopdf の0.10.0以降をご利用ください。

JavaEE開発モデル

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。

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.pdf.ApplicationPDFConverterAsync （以下 ApplicationPDFConverterAsync 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を推奨します。

実装例1：オプション指定なし

```
import java.io.File;

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


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

    // PDF出力処理
    final ApplicationPDFConverterAsync pdfConverter = new ApplicationPDFConverterAsync();
    File pdfFile = null;
    try {
        pdfFile = pdfConverter.createPDFapplicationId, userDataId, userCd);

        : (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();
        }
    }
```

実装例2：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.ApplicationPDFConverterAsync;
import jp.co.intra_mart.foundation.forma.exception.FormaSystemException;

   // 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
   final String userCd        = "XX"; // ユーザコード

   // wkhtmltopdfオプション
   // forma-pdf-config.xmlの設定内容を取得
   final List<String> options = PDFConfigOption.getList();
   // リンク無効を追加
   options.add("--disable-external-links");
   // 印刷モードを追加
   options.add("--print-media-type");

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

- PDFファイルの出力方向を標準の縦方向から横方向に変更する場合、オプション指定で以下の内容を記述してください。

```
options.add("--orientation");
options.add("Landscape");
```

> **コラム**
>
> オプションに設定値がある場合は、上記のようにoptionsに分けて設定してください。

> **コラム**
>
> ロードバランシング環境にて、改善モジュールを利用している場合は、APIの引数としてオプションを指定する必要があります。
> オプションの設定値については、以下を参照ください。
>
> - FAQ詳細ページ

実装例3：オプション指定あり、任意のリクエストパラメータ指定あり

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

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

   // 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
   final String userCd        = "XX"; // ユーザコード

   // 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");

   // 任意のリクエストパラメータ
   final Map<String, Object> optionalParams = new HashMap<String, Object>();
   map.put("任意KEY1", "任意パラメータ1");
   map.put("任意KEY2", "任意パラメータ2");

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

      : (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();
      }
   }
```

コラム

PDF出力対象画面に対し、任意のリクエストパラメータを設定できます。
前処理に任意のパラメータを受け渡す場合は、「*前処理* 」を参照してください。

スクリプト開発モデル

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。

- ApplicationPDFConverterAsync 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を推奨します。

実装例1：オプション指定なし

```
// 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
var userCd        = "XX"; // ユーザコード

// PDF出力処理
var pdfConverter = new FRApplicationPDFConverterAsync();
var result = pdfConverter.createPDF(applicationId, insertId, userCd);
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();
}
```

実装例2：オプション指定あり

```
// 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
var userCd        = "XX"; // ユーザコード

// wkhtmltopdfオプション
var options = [];
// forma-pdf-config.xmlの設定内容を取得
options = FRApplicationPDFConverterAsync.getOptions()
// リンク無効を追加
options.push("--disable-external-links");
// 印刷モードを追加
options.push("--print-media-type");

// PDF出力処理
var pdfConverter = new FRApplicationPDFConverterAsync();
var result = pdfConverter.createPDF(applicationId, insertId, userCd, options);
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();
}
```

- PDFファイルの出力方向を標準の縦方向から横方向に変更する場合、オプション指定で以下の内容を記述してください。

```
options.push("--orientation");
options.push("Landscape");
```

コラム

オプションに設定値がある場合は、上記のように分けてoptionsに分けて設定してください。

コラム

ロードバランシング環境にて、改善モジュールを利用している場合は、APIの引数としてオプションを指定する必要があります。
オプションの設定値については、以下を参照ください。

- FAQ詳細ページ

実装例3：オプション指定あり、任意のリクエストパラメータ指定あり

```
// 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
var userCd        = "XX"; // ユーザコード

// 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");

// 任意のリクエストパラメータ
var optionalParams = {};
optionalParams.sampleKEY1 = "任意パラメータ1";
optionalParams.sampleKEY2 = "任意パラメータ2";

// PDF出力処理
var pdfConverter = new FRApplicationPDFConverterAsync();
var result = pdfConverter.createPDF(applicationId, insertId, userCd, options, optionalParams);
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();
}
```

コラム

PDF出力対象画面に対し、任意のリクエストパラメータを設定できます。
前処理に任意のパラメータを受け渡す場合は、「 *前処理* 」を参照してください。

注意

wkhtmltopdf は2023年1月にアーカイブされました。
そのため、wkhtmltopdf の運用は非推奨となりました。

wkhtmltopdf の代替方法は以下のURLを参照してください。
https://product.intra-mart.support/hc/ja/articles/36748979822873

コラム

出力されたPDFファイルに埋め込まれるフォントは、「wkhtmltopdf」がインストールされている OS に依存します。
そのため、参照画面をブラウザで閲覧する場合とPDFで閲覧する場合に見た目が異なる場合があります。

コラム

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、以下のオプションを付与しています。
また、forma-pdf-config.xmlで設定した内容は、APIを使用する場合は反映されません。
API利用時に個別に設定する必要があります。

- 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、helpコマンドを参照してください。

helpコマンド

```
$ wkhtmltopdf -H
```

コラム

PDF出力が正常に実行できた場合、PDFファイルはWebアーカイブディレクトリの WEB-INF/work/product/forma/tmp/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、WEB-INF/work/product/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。WEB-INF/work/product/forma/tmp/print/ にゴミが残っている場合は、削除してください。

テンプレートの格納先

入力チェック

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。

```
%CONTEXT_PATH%/WEB-INF/jssp/src/sample/forma/template/input_validation.js
```

後処理プログラム

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からダウンロードしてください。

im_forma_sample-src.zip

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。

```
%CONTEXT_PATH%/WEB-INF/jssp/src/sample/forma/template/post_process.js
```

採番プログラム

採番ルール定義機能では、「番号接頭語」を任意のプログラムにより、作成することができます。
プログラムを実行するタイミングは、「画面アクセス毎」と「登録処理毎」があり、画面アイテム「採番」のプロパティで指定されます。

採番プログラムの実装

JavaEE開発モデル

JavaEE開発モデルにおいて、採番プログラムを実装する手順を示します。

実装規約

JavaEE開発モデルにおいて、採番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.AutoNoControllCodeCreator を継承すること
- 以下のメソッドを実装すること
  createControllCode(final UniqueNoInfoModel model, final Map<String, Object> sendParam) メソッド

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。

jp.co.intra_mart.foundation.forma.model.autono.UniqueNoInfoModel

採番ルール定義情報です。

メソッドの概要

メソッド	説明
long getIncrementalVal()	採番増分値を返却します。
int getDigits()	採番桁数を返却します。
String getSuffix()	番号接尾語を返却します。
String getUniqueNo()	採番定義番号を返却します。

sendParamマップ

採番方法が、画面アクセス毎の場合は空オブジェクトを保持します。

```
Map<String, Object> sendParam : 送信パラメータマップ
```

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。
画面アイテム「明細テーブル」、「グリッドテーブル」を除く入力アイテムのキーはフィールド識別ID です。
画面アイテム「明細テーブル」、「グリッドテーブル」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」-「リクエストパラメータ」を参照してください。

- 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

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	システムタイムゾーンにおける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： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。

- 画面アイテム「明細テーブル」、「グリッドテーブル」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 Map の List です。
  各列のデータ型は列タイプに依存します。

返却値とエラー処理

返却値に、番号接頭語を戻します。
処理中にエラーが発生した場合は、 Exception を throw してください。

トランザクションの制御

採番プログラムは採番方法が登録処理毎の場合、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。
採番方法を登録処理毎で利用する場合は、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ください。

実装例

```
1  public class SampleAutoNoCode extends AutoNoControllCodeCreator {
2
3     /**
4      * 登録処理を行った場合に実行されます。<BR>
5      * @param model 採番マスタ情報
6      * @param sendParam 送信パラメータ
7      * @return 番号接頭語
8      * @throws Exception 例外が発生
9      */
10    @Override
11    public String createControllCode(final UniqueNoInfoModel model, final Map<String, Object> sendParam) throws Exception {
12       return "%番号接頭語%";
13    }
14
15 }
```

スクリプト開発モデル

スクリプト開発モデルにおいて、採番プログラムを実装する手順を示します。

実装規約

スクリプト開発モデルにおいて、採番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。

- 以下のメソッドを実装すること
  createControllCode(autoNoParam , sendParam) メソッド

パラメータ

autoNoParamオブジェクト

採番ルール定義情報です。

```
autoNoParam : 採番マスタ情報オブジェクト
 └uniqueNo : 採番定義番号
```

sendParamマップ

採番方法が、画面アクセス毎の場合は空のオブジェクトを保持します。

```
Map<String, Object> sendParam : 送信パラメータマップ
```

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。
画面アイテム「明細テーブル」、「グリッドテーブル」を除く入力アイテムのキーはフィールド識別ID です。
画面アイテム「明細テーブル」、「グリッドテーブル」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」-「リクエストパラメータ」を参照してください。

- 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	システムタイムゾーンにおける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： 画面アイテム「関数」、「一覧選択」は取得値の設定により画面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画面アイテム（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。

- 画面アイテム「明細テーブル」、「グリッドテーブル」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 Map の List です。
各列のデータ型は列タイプに依存します。

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。
 └ data：番号接頭語
```

トランザクションの制御

採番プログラムは採番方法が登録処理毎の場合、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。
採番方法を登録処理毎で利用する場合は、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ください。

実装例

```
function createControllCode(autoNoParam, sendParam) {
    return {"error" : false, data : "%採番接頭語%"};
}
```

ワークフロー案件処理API

APIの位置づけ

IM-FormaDesigner for Accel Platform では、ユーザによる画面操作を経由することなく、プログラム上から案件の処理を行うための API を提供しています。

API を利用することで、以下のようなことが実現できます。

- 業務システムから申請用データを生成し、ジョブ・プログラムから全ユーザに向けて一括で申請処理を行う
- 特定のノードに到達した際に、条件に合致する場合は到達処理・ユーザプログラムにて当該ノードを自動で処理する

API のインタフェースの詳細については「IM-FormaDesigner for Accel Platform APIドキュメント」を参照してください

本ドキュメントでは、API の基本的な利用方法と注意事項について説明します。

> **注意**
>
> IM-Workflow の API も提供されていますが、IM-FormaDesigner for Accel Platform と連携したフロー定義に関しては必ず IM-FormaDesigner のAPIを利用してください。
> IM-FormaDesigner の API を経由することで、IM-FormaDesigner が IM-Workflow のコンテンツ定義に設定している後処理（アクション処理ユーザプログラム）に対して入力値の情報を連動することが可能です。IM-Workflow のAPIを直接利用した場合は、入力値が後処理に正しく連動されず、エラーが発生します。

> **注意**
>
> また、IM-BIS と連携したフロー定義に関しては、同様に IM-BIS のAPIを利用してください。
>
> IM-BIS の API については「IM-BIS for Accel Platform APIドキュメント」を参照してください。

JavaEE開発モデル

ここでは、ワークフロー案件処理API（JavaEE開発モデル）の基本的な利用方法を説明します。
APIのインタフェース情報については「IM-FormaDesigner for Accel Platform APIドキュメント（JavaEE開発モデル）」を参照してください。

> **注意**
>
> この章では、IM-FormaDesigner の API を利用した場合のサンプルコードを掲載しています。

申請処理の実装

パラメータ

申請処理APIを実行するには、ワークフローデータおよびアプリケーションデータの2つのパラメータが必要です。

- パラメータ（ワークフローデータ）

  ワークフローの案件を作成するための情報です。下記は必須項目です。

パラメータ（ワークフローデータ）

パラメータ	説明
フローID	申請時に利用するフロー定義のIDを指定します。
案件名	作成する案件のタイトルを指定します。
申請基準日	ワークフローの処理を行う際の基準日を表す日付文字列を指定します。
申請実行者コード	申請処理を実行するユーザのユーザコードを指定します。 通常は、権限者と同一のユーザを指定しますが、代理申請の場合には代理先ユーザのユーザコードを指定します。
申請権限者コード	ワークフローの案件の申請者として記録されるユーザコードを指定します。 通常は、実行者と同一のユーザを指定しますが、代理申請の場合には代理元ユーザのユーザコードを指定します。

```
// パラメータ（ワークフローデータ）の作成
final ApplyParam applyParam = new ApplyParam();
applyParam.setFlowId("SampleFlowId");
applyParam.setMatterName("サンプル案件");
applyParam.setApplyBaseDate("2015/02/14"); // yyyy/MM/ddの形式で指定します。
applyParam.setApplyExecuteUserCode("ueda");
applyParam.setApplyAuthUserCode("aoyagi");
```

- パラメータ（アプリケーションデータ）

ワークフローの案件にひもづく業務データです。以下の3種類が存在します。

パラメータ（アプリケーションデータ）

種類	説明
入力アイテムデータ	単項目の画面アイテムに入力するデータを指定します。
テーブルアイテムデータ	明細項目の画面アイテムに入力するデータを指定します。
ファイルアップロードアイテムデータ	画面アイテム「ファイルアップロード」に添付するファイル情報を指定します。

```
// パラメータ（アプリケーションデータ）の作成
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();
final Map<String, Object> items = new HashMap<String, Object>();
final FormaFileUpload[] files = new FormaFileUpload[n];
formaUserParam.put(StandardFormaUserParamKey.ITEMS, items);
formaUserParam.put(StandardFormaUserParamKey.FILES, files);
```

- パラメータ（アプリケーションデータ） — 入力アイテムデータ

フィールド識別IDをキーに各アイテムの入力値を作成していきます。
各アイテムのデータ型に応じてセットするデータ型が異なることに注意してください。

```
// パラメータ（アプリケーションデータ）の作成
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();
final Map<String, Object> items = new HashMap<String, Object>();

// フィールド識別IDをキーに入力値をセットします。
items.put("textbox1", "あいうえお"); // 文字列型のアイテムは、String型で指定します。
items.put("number1", 1000); // 数値型のアイテムは、int型またはNumber型で指定します。
items.put("calendar1", new Date()); // 日付型は、java.util.Date型で指定します。
items.put("timestamp1", new Date()); // タイムスタンプ型は、java.util.Date型で指定します。

// 入力アイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.put(StandardFormaUserParamKey.ITEMS, items);
```

コラム

複数選択可能な文字列型のアイテム（チェックボックス・リストボックスなど）では、カンマ区切りで複数要素を指定します。

```
items.put("checkbox1", "sample1,sample2,sample3");
```

- パラメータ（アプリケーションデータ） — テーブルアイテムデータ

各行ごとに、入力値を格納するjava.util.Mapを作成します。
Mapのキーに各列のフィールド識別IDを指定し、値に入力値を指定します。

```
// パラメータ（アプリケーションデータ）の作成
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();
final Map<String, Object> items = new HashMap<String, Object>();

// テーブルアイテムデータを格納するためのjava.util.Listを作成します。
// Listの1要素がテーブルの1行に相当します。
final List<Map<String, Object>> tb1 = new ArrayList<Map<String, Object>>();

// テーブルアイテム 1行目のデータ
// 各カラムの入力値は、java.util.Mapに格納します。
// Mapのキーは、各列のフィールド識別IDです。
final Map<String, Object> detail11 = new HashMap<String, Object>();
detail11.put("tb1_textbox1", "あいうえお");
detail11.put("tb1_number1", 1000);
detail11.put("tb1_calendar1", new Date());
detail11.put("tb1_timestamp1", new Date());
tb1.add(detail11);

// テーブルアイテム 2行目のデータ
final Map<String, Object> detail12 = new HashMap<String, Object>();
detail12.put("tb1_textbox1", "かきくけこ");
detail12.put("tb1_number1", 2000);
detail12.put("tb1_calendar1", new Date());
detail11.put("tb1_timestamp1", new Date());
tb1.add(detail12);

// テーブル識別IDをキーに各テーブルアイテムの情報を格納します。
items.put("tb1", tb1);

// テーブルアイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.put(StandardFormaUserParamKey.ITEMS, items);
```

- パラメータ（アプリケーションデータ） — ファイルアップロードアイテムデータ

  添付ファイル情報を格納するためのモデルFormaFileUploadを作成します。
  各ファイルの実体は、InputStreamで指定します。

```
// パラメータ（アプリケーションデータ）の作成
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();
InputStream in = null;
try {
    in = new FileInputStream(new File("C://Users/ogawas/Desktop/api.java"));
} catch (final IOException e) {
    e.printStackTrace();
}

// 添付するファイルごとにFormaFileUpload型のモデルを作成します。
final FormaFileUpload file1 = new FormaFileUpload();
file1.setFile(in); // 添付ファイルの実体をjava.io.InputStream型で指定します。
file1.setFileName("sampleFile.txt"); // 添付ファイルのファイル名を指定します。
file1.setUploadItemId("attach_fileupload_item1"); // アイテム識別IDを指定します。
file1.setNotes("サンプルのファイルです。");

// FormaFileUpload型の配列を作成します。
final FormaFileUpload[] files = new FormaFileUpload[] { file1 };

// ファイルアップロードアイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.put(StandardFormaUserParamKey.FILES, files);
```

申請処理を実装する

ワークフローデータ、アプリケーションデータのパラメータをそれぞれ指定した上で、申請処理APIを実行します。

```
// 申請処理APIのインスタンスを作成します。
final FormaApplyManager applyManager = new FormaApplyManager();

final ApplyParam applyParam = new ApplyParam();
final Map<String, Object> userParam = new HashMap<String, Object>();
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();
final Map<String, Object> items = new HashMap<String, Object>();

// パラメータ（ワークフローデータ）の作成
applyParam.setFlowId("SampleFlowId");
applyParam.setMatterName("サンプル案件");
applyParam.setApplyBaseDate("2015/02/14");
applyParam.setApplyExecuteUserCode("aoyagi");
applyParam.setApplyAuthUserCode("aoyagi");

// パラメータ（アプリケーションデータ）の作成
items.put("textbox1", "aiueotextbox");
items.put("number1", 1000);
formaUserParam.put(StandardFormaUserParamKey.ITEMS, items);

// 作成したパラメータを指定した上で、申請処理APIを実行します。
final ApplyResultModel applyResult = applyManager.apply(applyParam, userParam, formaUserParam);
```

戻り値

jp.co.intra_mart.foundation.workflow.application.model.ApplyResultModel 型のモデルとして、戻り値が返却されます。以下の情報が取得できます。

戻り値

パラメータ	説明
システム案件ID	作成したワークフローの案件を一意に識別できるID情報。 承認処理APIを実行する際には、指定する必要があります。
ユーザデータID	作成したアプリケーションデータを一意に識別できるID情報。
案件番号	ユーザが画面から案件を識別するためのID情報。

承認処理の実装

案件と処理ノードを特定するためのパラメータを指定して、承認処理APIのインスタンスを作成します。
作成した承認処理APIのインスタンスに、案件を処理するためのパラメータを指定して、実行します。

パラメータ

承認処理APIを実行するには、ワークフローデータおよびアプリケーションデータの2つのパラメータが必要です。

- パラメータ（ワークフローデータ）

  ワークフローの案件を処理する際のための情報です。下記は必須項目です。

  パラメータ（ワークフローデータ）

パラメータ	説明
処理実行者コード	承認処理を実行するユーザのユーザコードを指定します。 通常は、権限者と同一のユーザを指定しますが、代理承認の場合には代理先ユーザのユーザコードを指定します。
処理権限者コード	ワークフローの案件の承認者として記録されるユーザコードを指定します。 通常は、実行者と同一のユーザを指定しますが、代理承認の場合には代理元ユーザのユーザコードを指定します。

  ```
  // パラメータ（ワークフローデータ）の作成
  final ApproveParam approveParam = new ApproveParam();
  approveParam.setExecuteUserCode("ueda");
  approveParam.setAuthUserCode("aoyagi");
  ```

- パラメータ（アプリケーションデータ）

  ワークフローの案件にひもづく業務データです。パラメータの基本的なデータ構造は申請時と同じですが、承認時には追記する項目のみセットします。

```
// パラメータ（アプリケーションデータ）の作成
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();
final Map<String, Object> items = new HashMap<String, Object>();

// フィールド識別IDをキーに入力値をセットします。
items.put("timestamp1", new Date()); // 追記する項目のみセットします。

// 入力アイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.put(StandardFormaUserParamKey.ITEMS, items);
```

承認処理を実装する

承認処理に必要なパラメータを作成します。
案件と処理ノードを特定するためのパラメータを指定して、承認処理APIのインスタンスを作成します。
作成した承認処理APIのインスタンスにパラメータを指定して、実行します。

```
final ApproveParam approveParam = new ApproveParam();
final Map<String, Object> userParam = new HashMap<String, Object>();
final Map<FormaUserParamKey, Object> formaUserParam = new HashMap<FormaUserParamKey, Object>();

// パラメータ（ワークフローデータ）の作成
approveParam.setExecuteUserCode("ueda");
approveParam.setAuthUserCode("aoyagi");

// パラメータ（アプリケーションデータ）の作成
final Map<String, Object> items = new HashMap<String, Object>();
items.put("calendar1", new Date()); // 追記可能な項目にのみ値をセットします。
formaUserParam.put(StandardFormaUserParamKey.ITEMS, items);

// システム案件ID・ノードIDを指定して、承認処理APIのインスタンスを作成します。
final FormaProcessManager processManager = new FormaProcessManager("SampleSystemMatterId", "SampleNodeId");

// 作成したパラメータを指定した上で、承認処理APIを実行します。
processManager.approve(approveParam, userParam, formaUserParam);
```

戻り値

戻り値はありません。

- エラー時は jp.co.intra_mart.foundation.forma.exception.FormaApiException クラスがスローされます。

スクリプト開発モデル

ここでは、ワークフロー案件処理API（スクリプト開発モデル）の基本的な利用方法を説明します。
APIのインタフェース情報については「IM-FormaDesigner for Accel Platform APIドキュメント（スクリプト開発モデル）」を参照してください。

> **注意**
>
> この章では、IM-FormaDesigner の API を利用した場合のサンプルコードを掲載しています。

申請処理の実装

パラメータ

申請処理APIを実行するには、ワークフローデータおよびアプリケーションデータの2つのパラメータが必要です。

- パラメータ（ワークフローデータ）

  ワークフローの案件を作成するための情報です。下記は必須項目です。

パラメータ（ワークフローデータ）

パラメータ	説明
フローID	申請時に利用するフロー定義のIDを指定します。
案件名	作成する案件のタイトルを指定します。
申請基準日	ワークフローの処理を行う際の基準日を表す日付文字列を指定します。
申請実行者コード	申請処理を実行するユーザのユーザコードを指定します。 通常は、権限者と同一のユーザを指定しますが、代理申請の場合には代理先ユーザのユーザコードを指定します。
申請権限者コード	ワークフローの案件の申請者として記録されるユーザコードを指定します。 通常は、実行者と同一のユーザを指定しますが、代理申請の場合には代理元ユーザのユーザコードを指定します。

```
// パラメータ（ワークフローデータ）の作成
var applyParam = {};
applyParam.flowId = 'SampleFlowId';
applyParam.matterName = 'サンプル案件';
applyParam.applyBaseDate = '2015/02/14';
applyParam.applyExecuteUserCode = 'ueda';
applyParam.applyAuthUserCode = 'aoyagi';
```

- パラメータ（アプリケーションデータ）

ワークフローの案件にひもづく業務データです。以下の3種類が存在します。

パラメータ（アプリケーションデータ）

種類	説明
入力アイテムデータ	単項目の画面アイテムに入力するデータを指定します。
テーブルアイテムデータ	明細項目の画面アイテムに入力するデータを指定します。
ファイルアップロードアイテムデータ	画面アイテム「ファイルアップロード」に添付するファイル情報を指定します。

```
// パラメータ（アプリケーションデータ）の作成
var formaUserParam = {};
var items = {};
var files = [n];

formaUserParam.items = items;
formaUserParam.files = files;
```

- パラメータ（アプリケーションデータ） — 入力アイテムデータ

フィールド識別IDをキーに各アイテムの入力値を作成していきます。
各アイテムのデータ型に応じてセットするデータ型が異なることに注意してください。

```
// パラメータ（アプリケーションデータ）の作成
var formaUserParam = {};
var items = {};

// フィールド識別IDをキーに入力値をセットします。
items.textbox1 = 'あいうえお'; // 文字列型のアイテムは、String型で指定します。
items.number1 = 1000;// 数値型のアイテムは、Number型で指定します。
items.calendar1 = new Date(); // 日付型は、Date型で指定します。
items.timestamp1 = new Date(); // タイムスタンプ型は、Date型で指定します。

// 入力アイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.items = items;
```

> **コラム**
>
> 複数選択可能な文字列型のアイテム（チェックボックス・リストボックスなど）では、カンマ区切りで複数要素を指定します。
>
> ```
> items.checkbox1 = 'sample1,sample2,sample3';
> ```

- パラメータ（アプリケーションデータ） — テーブルアイテムデータ

  各行ごとに、入力値を格納するオブジェクトを作成します。
  オブジェクトのプロパティに各列のフィールド識別IDを指定し、値に入力値を指定します。

```
// パラメータ（アプリケーションデータ）の作成
var formaUserParam = {};
var items = {};

// テーブルアイテムデータを格納するための配列を作成します。
// 配列の1要素がテーブルの1行に相当します。
var tb1 = [];

// テーブルアイテム 1行目のデータ
// 各カラムの入力値は、オブジェクトに格納します。
// オブジェクトのプロパティは、各列のフィールド識別IDです。
var detail11 = {};
detail11.tb1_textbox1 = 'あいうえお';
detail11.tb1_number1 = 1000;
detail11.tb1_calendar1 = new Date();
detail11.tb1_timestamp1 = new Date();
tb1.push(detail11);

// テーブルアイテム 2行目のデータ
var detail12 = {};
detail12.tb1_textbox1 = 'かきくけこ';
detail12.tb1_number1 = 2000;
detail12.tb1_calendar1 = new Date();
detail12.tb1_timestamp1 = new Date();
tb1.push(detail12);

// テーブル識別IDをキーに各テーブルアイテムの情報を格納します。
items.tb1 = tb1;

// テーブルアイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.items = items;
```

- パラメータ（アプリケーションデータ） — ファイルアップロードアイテムデータ

  添付ファイル情報を格納するためのオブジェクトを作成します。
  ファイルの実体は、Fileオブジェクトで指定します。

```
// パラメータ（アプリケーションデータ）の作成
var formaUserParam = {};
var items = {};

// 添付するファイルごとにオブジェクトを作成します。
var file1 = {};
file1.file = new File('C://sample/sampleFile.txt'); // 添付ファイルの実体をFileオブジェクトで指定します。
file1.uploadItemId = 'attach_fileupload_item1'; // アイテム識別IDを指定します。
file1.notes = 'サンプルのファイルです。';

// ファイル情報のオブジェクトを格納する配列を作成します。
var files = [file1];

// ファイルアップロードアイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.files = files;
```

申請処理を実装する

ワークフローデータ・アプリケーションデータのパラメータをそれぞれ指定した上で、申請処理APIを実行します。

```
// 申請処理APIのインスタンスを作成します。
var applyManager = new FormaApplyManager();

var applyParam = {};
var userParam = {};
var formaUserParam = {};

var items = {};

// パラメータ（ワークフローデータ）の作成
applyParam.flowId = 'SampleFlowId';
applyParam.matterName = 'サンプル案件';
applyParam.applyBaseDate = '2015/02/14';
applyParam.applyExecuteUserCode = 'aoyagi';
applyParam.applyAuthUserCode = 'aoyagi';

// パラメータ（アプリケーションデータ）の作成
items.textbox1 = 'あいうえお';
items.number1 = 1000;
formaUserParam.items = items;

// 作成したパラメータを指定した上で、申請処理APIを実行します。
var applyResult = applyManager.apply(applyParam, userParam, formaUserParam);
```

戻り値

処理結果に加えて、以下の情報が取得できます。

戻り値

パラメータ	説明
システム案件ID	作成したワークフローの案件を一意に識別できるID情報。 承認処理APIを実行する際には、指定する必要があります。
ユーザデータID	作成したアプリケーションデータを一意に識別できるID情報。
案件番号	ユーザが画面から案件を識別するためのID情報。

承認処理の実装

案件と処理ノードを特定するためのパラメータを指定して、承認処理APIのインスタンスを作成します。
作成した承認処理APIのインスタンスに、案件を処理するためのパラメータを指定して、実行します。

パラメータ

承認処理APIを実行するには、ワークフローデータおよびアプリケーションデータの2つのパラメータが必要です。

- パラメータ（ワークフローデータ）

  ワークフローの案件を処理する際のための情報です。下記は必須項目です。

  パラメータ（ワークフローデータ）

パラメータ	説明
処理実行者コード	承認処理を実行するユーザのユーザコードを指定します。 通常は、権限者と同一のユーザを指定しますが、代理承認の場合には代理先ユーザのユーザコードを指定します。
処理権限者コード	ワークフローの案件の承認者として記録されるユーザコードを指定します。 通常は、実行者と同一のユーザを指定しますが、代理承認の場合には代理元ユーザのユーザコードを指定します。

```
// パラメータ（ワークフローデータ）の作成
var approveParam = {};
approveParam.applyExecuteUserCode = 'ueda';
approveParam.applyAuthUserCode = 'aoyagi';
```

- パラメータ（アプリケーションデータ）

  ワークフローの案件にひもづく業務データです。パラメータの基本的なデータ構造は申請時と同じですが、承認時には追記する項目のみセットします。

```
// パラメータ（アプリケーションデータ）の作成
var formaUserParam = {};
var items = {};

// フィールド識別IDをキーに入力値をセットします。
items.timestamp1 = new Date(); // 追記する項目のみセットします。

// 入力アイテムデータをアプリケーションデータのパラメータにセットします。
formaUserParam.items = items;
```

承認処理を実装する

承認処理に必要なパラメータを作成します。
案件と処理ノードを特定するためのパラメータを指定して、承認処理APIのインスタンスを作成します。
作成した承認処理APIのインスタンスにパラメータを指定して、実行します。

```
var approveParam = {};
var userParam = {};
var formaUserParam = {};

// パラメータ（ワークフローデータ）の作成
approveParam.executeUserCode = 'ueda';
approveParam.authUserCode = 'aoyagi';

// パラメータ（アプリケーションデータ）の作成
var items = {};
items.calendar1 = new Date(); // 追記可能な項目にのみ値をセットします。
formaUserParam.items = items;

// システム案件ID・ノードIDを指定して、承認処理APIのインスタンスを作成します。
var processManager = new FormaProcessManager('SampleSystemMatterId', 'SampleNodeId');

// 作成したパラメータを指定した上で、承認処理APIを実行します。
processManager.approve(approveParam, userParam, formaUserParam);
```

戻り値

処理結果のみ取得できます。

注意事項

画面仕様との差異

ユーザが IM-FormaDesigner の画面から操作を行う場合と異なり、ワークフロー案件処理APIから申請・処理を行った場合は、以下のチェックが実行されません。

- ワークフローの権限チェック：申請実行者・申請権限者、処理実行者・処理権限者に対する権限チェックは実行されない
- アプリケーションデータに対する入力チェック：アイテムのプロパティと入力チェック・ユーザプログラムに設定されている入力チェックは実行されない

トランザクション管理

ワークフロー案件処理APIについてはAPI内部処理にて独自にトランザクション管理を行っているため、呼び出し側のユーザプログラムにてトランザクションを管理することはできません。

業務データの更新

現状、以下の API を実行した場合は、FormaUserParam パラメータを指定してもアプリケーションデータは更新されません。

IM-FormaDesigner	▪ FormaProcessManager.deny ▪ FormaProcessManager.discontinue ▪ FormaProcessManager.reserve ▪ FormaProcessManager.reserveCancel ▪ FormaProcessManager.sendBack
IM-BIS	▪ BisProcessManager.deny ▪ BisProcessManager.discontinue ▪ BisProcessManager.reserve ▪ BisProcessManager.reserveCancel ▪ BisProcessManager.sendBack

注意

- ワークフロー案件処理APIを利用して申請を行う場合、パラメータ（アプリケーションデータ）に指定できるアイテムは、申請ノードに設定されている画面遷移のフォームに含まれるものに限定されます。
- ワークフロー案件処理APIを利用して承認を行う場合、パラメータ（アプリケーションデータ）に指定できるアイテムは、承認ノードに設定されている画面遷移のフォームに含まれるものに限定されます。

ワークフロー案件処理 Web API

全体の作業の流れ

ここでは、前章で説明したワークフロー案件処理APIをHTTPベースで実行できるWeb APIの利用方法を説明します。
Web APIを利用することで、外部システムから案件の処理を行うことが可能です。

ワークフロー案件処理 Web API を実行するためには、以下の作業を行う必要があります。

- APIの仕様を確認する
  intra-mart Accel Platform 上にデプロイされたWeb APIドキュメントと本章の補足説明からWeb APIの仕様を確認します。
- 認可設定を行う
  Web APIを実行するユーザに対して実行権限を付与する設定を行います。
- 認証設定を行う
  Web APIを実行するユーザが intra-mart Accel Platform の認証を通過できるように設定を行います。
- クライアントアプリケーションを実装する
  Web APIの実行に必要な設定を行った上で、実際にパラメータを作成しHTTPリクエストを送信します。

APIの仕様を確認する

ここでは、ワークフロー案件処理Web APIの仕様が掲載されているAPIドキュメントへのアクセス方法を説明します。
また、APIドキュメントで不足している情報については本章で補足説明します。

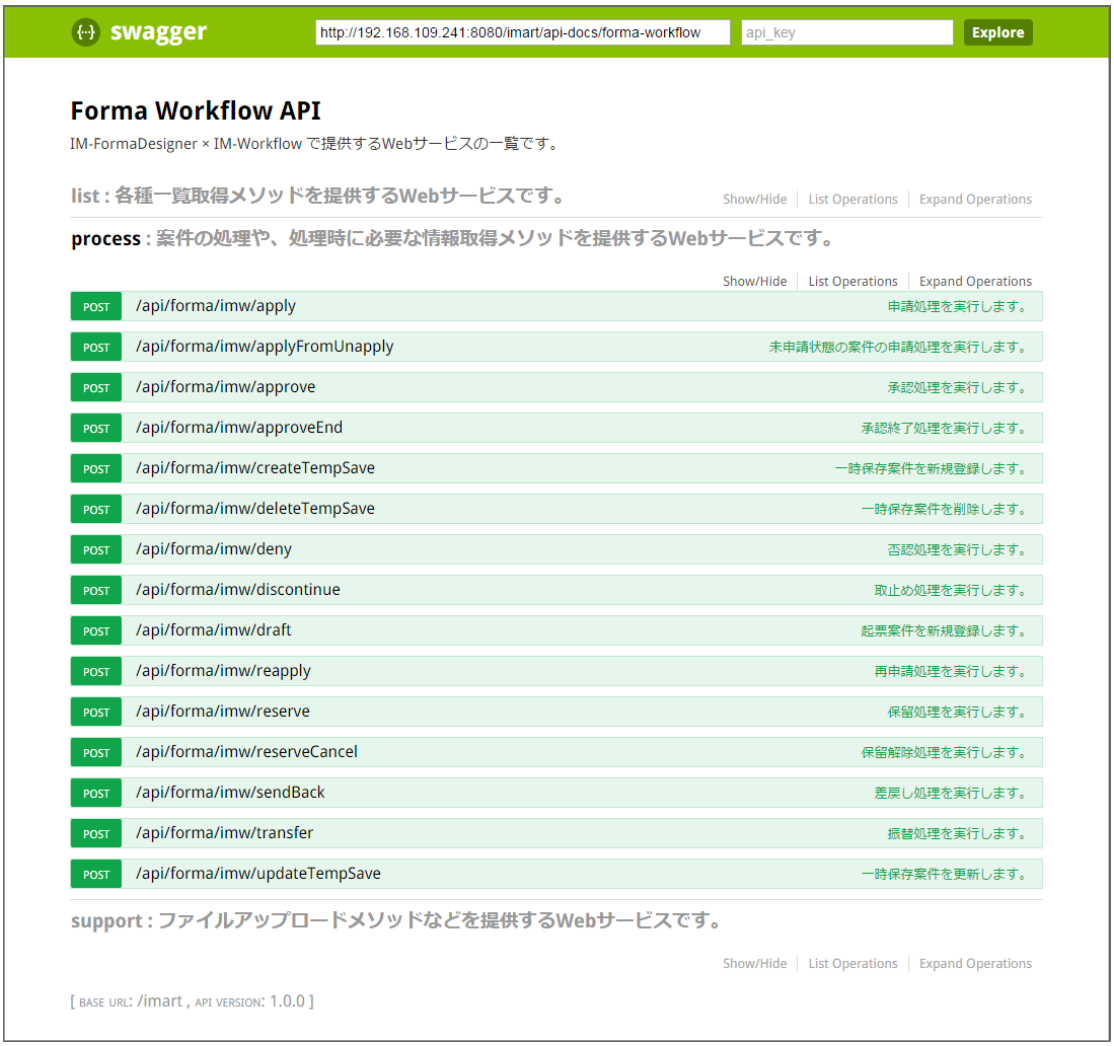
Web上からAPI仕様を参照し実行する

ワークフロー案件処理Web APIでは、Swagger Specificationを利用して、APIドキュメントを intra-mart Accel Platform 上のWeb アプリケーションとして提供しています。
Swagger Specificationでは、APIの仕様の確認だけでなく、画面上からパラメータ情報を入力することでWeb APIを実行することが可能です。

Swagger Specificationを表示する

Swagger Specificationから、提供されているWeb APIのURL一覧と各APIの実行に必要なリクエスト情報を確認することができます。

1. Swagger Specificationを利用するための intra-mart Accel Platform の環境を構築します。
   - 以下のモジュールをインストールした環境を構築してください。
     - IM-FormaDesigner for Accel Platform
   - IM-BIS ワークフロー案件処理Web APIについては、以下のモジュールをインストールした環境を構築してください。
     - IM-BIS for Accel Platform
2. メニューからSwagger Specificationへアクセスします。
   - IM-FormaDesigner ワークフロー案件処理Web API
     - 「サイトマップ」-「Forma開発者」-「Webサービス APIドキュメント」-「ワークフロー」をクリックし、Swagger Specificationを表示します。
   - IM-BIS ワークフロー案件処理Web API
     - 「サイトマップ」-「IM-BIS開発者」-「Webサービス APIドキュメント」-「ワークフロー」をクリックし、Swagger Specificationを表示します。

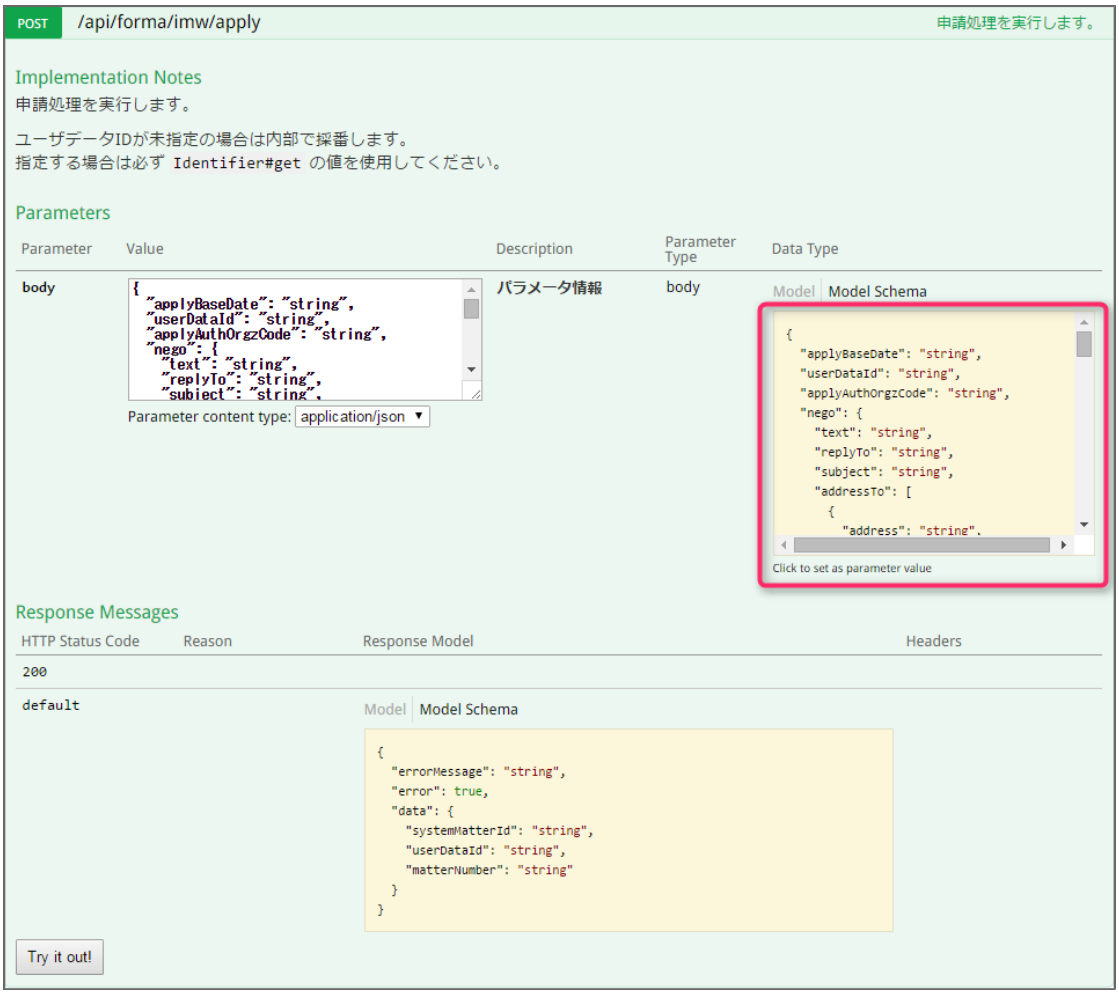


Swagger SpecificationからWeb APIを実行する

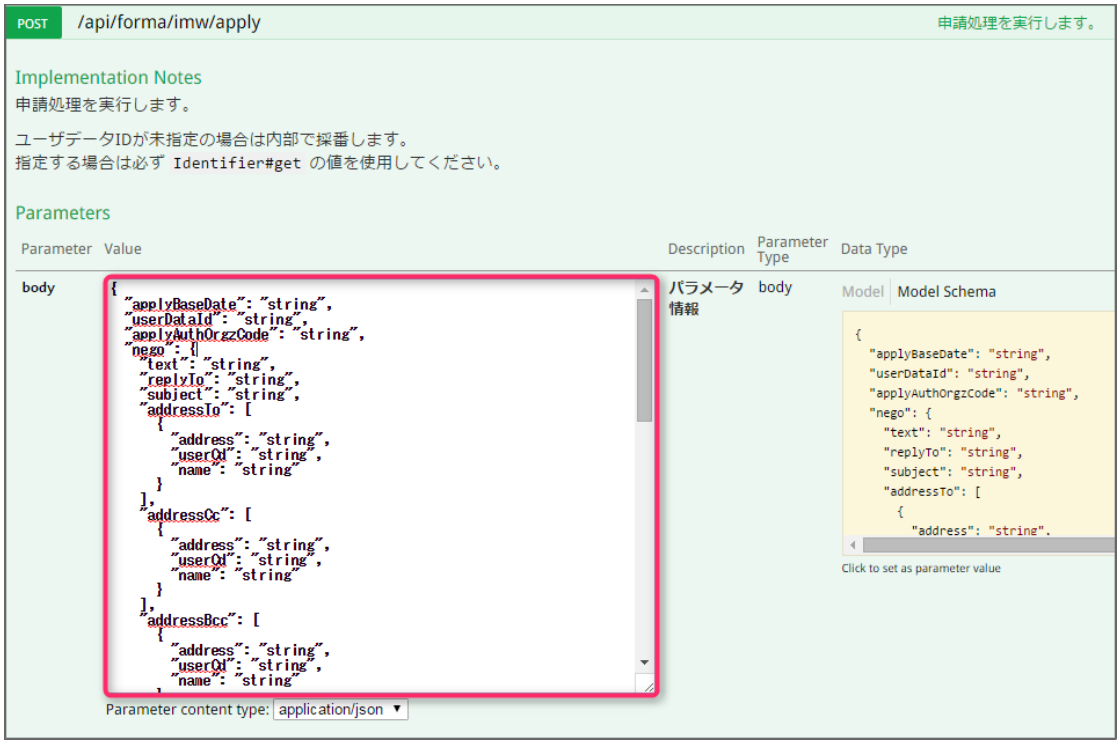
Swagger Specificationの画面に表示されているリクエスト情報のひな形を編集し、実際にWeb APIを実行することが可能です。

1. URLの一覧から実行したAPIを押下します。
   - APIのURLを押下すると、APIの詳細ページが表示されます。

2. 表示されたAPIの詳細画面からパラメータ情報を押下します。
   - パラメータ情報のJSON Schemaの部分を押下すると、Valueフィールドにパラメータ情報のひな形のJSONが編集可能な状態で表示されます。



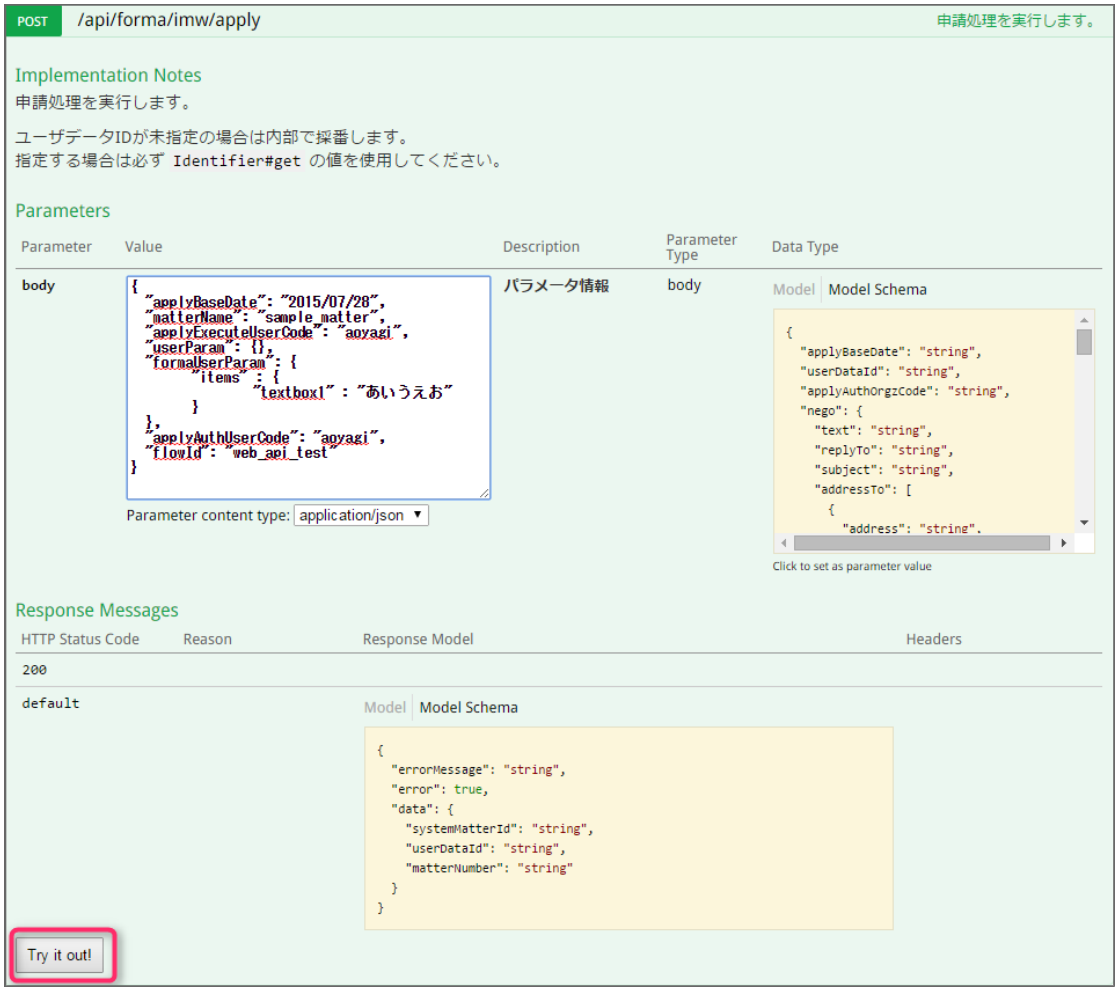
3. 表示されたパラメータ情報のひな形を編集します。



- ひな形のJSONを編集し、案件の処理が実行できるパラメータを作成します。
  - パラメータ例

```
{
 "applyBaseDate": "2015/07/28",
 "matterName": "sample_matter",
 "applyExecuteUserCode": "aoyagi",
 "userParam": {},
 "formaUserParam": {
  "items" : {
    "textbox1" : "あいうえお"
  }
 },
 "applyAuthUserCode": "aoyagi",
 "flowId": "web_api_test"
}
```

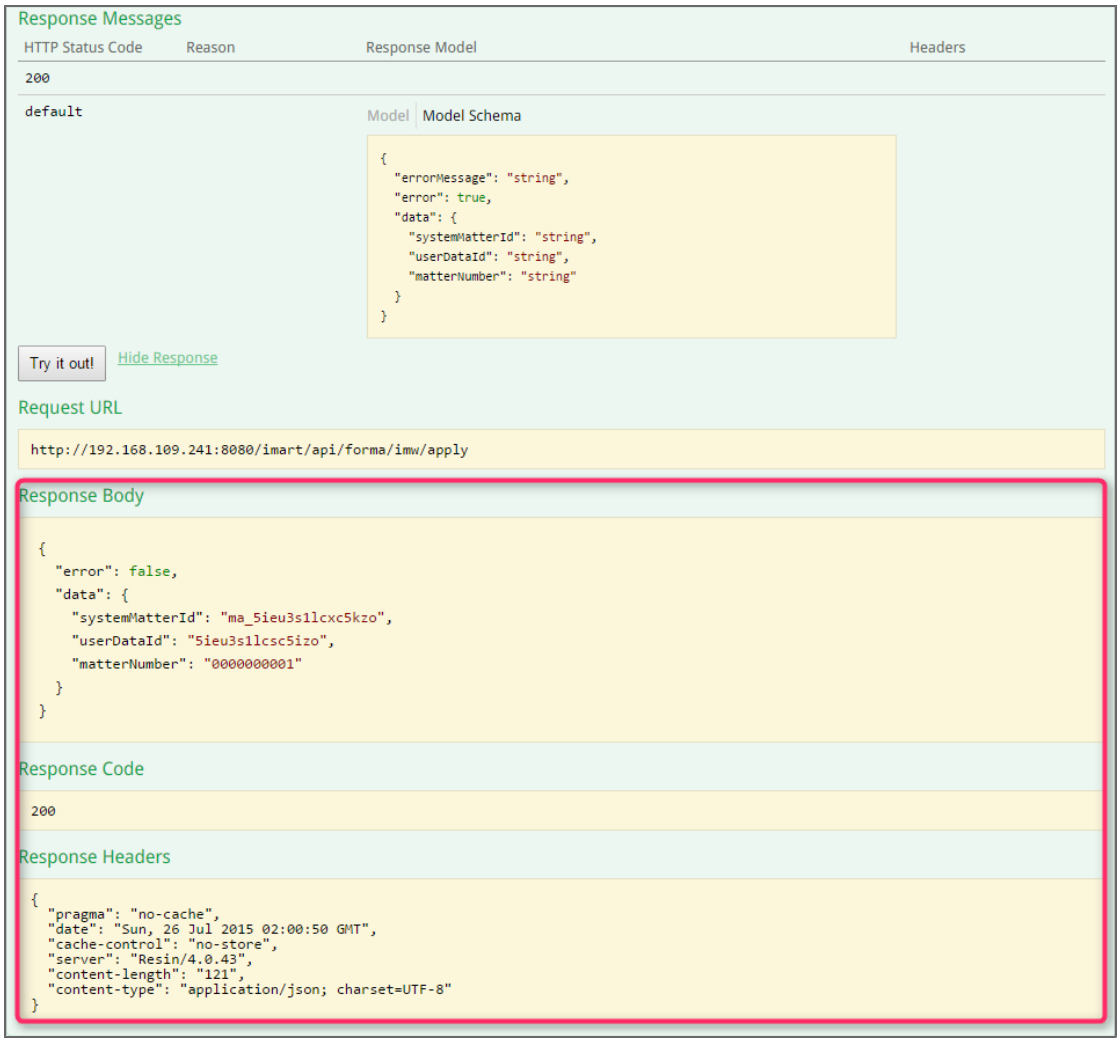
4. Web APIの実行ボタンを押下します。
   - 必要なパラメータを入力した上で「Try it out!」ボタンを押下します。



5. Web APIの実行結果を確認します。
   - Web APIの実行に成功した場合は、Response情報が表示されます。

Response情報

Response Headers	HTTP通信のレスポンスヘッダとしてメタ情報が表示されます。
Response Code	HTTP通信のレスポンスコードが表示されます。 APIの呼び出しに成功した場合は 200 が返却されます。 認可制御でエラーになった場合は 403 が返却されます。
Response Body	APIの実行結果の詳細が表示されます。



注意

Swagger SpecificationからWeb APIを実行した場合でも実際の案件処理が動作します。
本番環境では実施しないでください。

注意

Web APIの認証ユーザはSwagger Specificationの画面を操作しているログインユーザが設定されます。

提供APIのURL一覧

提供されるWeb APIは以下のとおりです。

IM-FormaDesigner

各種一覧取得メソッドを提供するWebサービス

メソッド	パス	説明
POST	/api/forma/imw/selectUnprocessActvMatterNodeList	未処理ノード一覧を取得します。
POST	/api/forma/imw/selectUnprocessActvMatterNodeListCount	未処理ノード一覧件数を取得します。
POST	/api/forma/imw/selectProcessedActvMatterList	処理済一覧（未完了）を取得します。
POST	/api/forma/imw/selectProcessedActvMatterListCount	処理済一覧（未完了）件数を取得します。
POST	/api/forma/imw/selectRefActvMatterList	参照一覧（未完了）を取得します。
POST	/api/forma/imw/selectRefActvMatterListCount	参照一覧（未完了）件数を取得します。
POST	/api/forma/imw/selectRefActvMatterAdminHandleLevelList	参照一覧（未完了）管理マネージャ（操作レベルを含む）を取得します。

メソッド	パス	説明
POST	/api/forma/imw/selectRefActvMatterAdminHandleLevelListCount	参照一覧（未完了）管理マネージャ（操作レベルを含む）件数を取得します。

案件の処理や、処理時に必要な情報取得メソッドを提供するWebサービス

メソッド	パス	説明
POST	/api/forma/imw/apply	申請処理を実行します。
POST	/api/forma/imw/applyFromUnapply	未申請状態の案件の申請処理を実行します。
POST	/api/forma/imw/approve	承認処理を実行します。
POST	/api/forma/imw/approveEnd	承認終了処理を実行します。
POST	/api/forma/imw/createTempSave	一時保存案件を新規登録します。 申請・起票前に一時保存を行った案件が対象です。
POST	/api/forma/imw/updateTempSave	一時保存案件を更新します。 申請・起票前に一時保存を行った案件が対象です。
POST	/api/forma/imw/deleteTempSave	一時保存案件を削除します。 申請・起票前に一時保存を行った案件が対象です。
POST	/api/forma/imw/deny	否認処理を実行します。
POST	/api/forma/imw/discontinue	取止め処理を実行します。
POST	/api/forma/imw/draft	起票案件を新規登録します。
POST	/api/forma/imw/fileupload	ファイルをアップロードします。
POST	/api/forma/imw/reapply	再申請処理を実行します。
POST	/api/forma/imw/reserve	保留処理を実行します。
POST	/api/forma/imw/reserveCancel	保留解除処理を実行します。
POST	/api/forma/imw/pullBack	引戻し処理を実行します。
POST	/api/forma/imw/sendBack	差戻し処理を実行します。
POST	/api/forma/imw/transfer	振替処理を実行します。

ファイルアップロードメソッドなどを提供するWebサービス

メソッド	パス	説明
POST	/api/forma/imw/fileupload	ファイルアップロード用のWeb API

IM-BIS

各種一覧取得メソッドを提供するWebサービス

メソッド	パス	説明
POST	/api/bis/imw/selectUnprocessActvMatterNodeList	未処理ノード一覧を取得します。
POST	/api/bis/imw/selectUnprocessActvMatterNodeListCount	未処理ノード一覧件数を取得します。
POST	/api/bis/imw/selectProcessedActvMatterList	処理済一覧（未完了）を取得します。
POST	/api/bis/imw/selectProcessedActvMatterListCount	処理済一覧（未完了）件数を取得します。
POST	/api/bis/imw/selectRefActvMatterList	参照一覧（未完了）を取得します。
POST	/api/bis/imw/selectRefActvMatterListCount	参照一覧（未完了）件数を取得します。
POST	/api/bis/imw/selectRefActvMatterAdminHandleLevelList	参照一覧（未完了）管理マネージャ（操作レベルを含む）を取得します。
POST	/api/bis/imw/selectRefActvMatterAdminHandleLevelListCount	参照一覧（未完了）管理マネージャ（操作レベルを含む）件数を取得します。

案件の処理や、処理時に必要な情報取得メソッドを提供するWebサービス

メソッド	パス	説明
POST	/api/bis/imw/apply	申請処理を実行します。
POST	/api/bis/imw/applyFromUnapply	未申請状態の案件の申請処理を実行します。
POST	/api/bis/imw/approve	承認処理を実行します。
POST	/api/bis/imw/approveEnd	承認終了処理を実行します。
POST	/api/bis/imw/createTempSave	一時保存案件を新規登録します。 申請・起票前に一時保存を行った案件が対象です。
POST	/api/bis/imw/updateTempSave	一時保存案件を更新します。 申請・起票前に一時保存を行った案件が対象です。
POST	/api/bis/imw/deleteTempSave	一時保存案件を削除します。 申請・起票前に一時保存を行った案件が対象です。
POST	/api/bis/imw/deny	否認処理を実行します。
POST	/api/bis/imw/discontinue	取止め処理を実行します。
POST	/api/bis/imw/draft	起票案件を新規登録します。
POST	/api/bis/imw/fileupload	ファイルをアップロードします。
POST	/api/bis/imw/reapply	再申請処理を実行します。
POST	/api/bis/imw/reserve	保留処理を実行します。
POST	/api/bis/imw/reserveCancel	保留解除処理を実行します。
POST	/api/bis/imw/pullBack	引戻し処理を実行します。
POST	/api/bis/imw/sendBack	差戻し処理を実行します。
POST	/api/bis/imw/transfer	振替処理を実行します。

ファイルアップロードメソッドなどを提供するWebサービス

メソッド	パス	説明
POST	/api/bis/imw/fileupload	ファイルアップロード用のWeb API

パラメータ情報の仕様を確認する

ワークフロー案件処理APIと同様に、パラメータはワークフローデータ・アプリケーションデータの2つが必要です。
パラメータは、リクエストボディからJSON形式で送信します。

ワークフローデータ

パラメータ（ワークフローデータ）については、Swagger Specificationのスキーマ情報から確認することができます。
パラメータの各プロパティについては、ワークフロー案件処理API（スクリプト開発モデル）と同様です。詳細は「IM-FormaDesigner for Accel Platform APIドキュメント（スクリプト開発モデル）」を参照してください。

アプリケーションデータ

Swagger Specificationのスキーマ情報ではMapのデータ構造を表現できないため、パラメータ（アプリケーションデータ）については本ドキュメントにてインタフェースを説明します。

パラメータ（アプリケーションデータ）

基本的には、ワークフロー案件処理API（スクリプト開発モデル）のパラメータ（アプリケーションデータ）のオブジェクトをJSONにフォーマットして指定します。

ただし、以下のデータ型についてはJSONに直接指定できないため、ワークフロー案件処理Web API独自の指定方法を定めています。

- 日付型
- タイムスタンプ型
- ファイル

アプリケーションデータ（入力アイテムデータ）

基本的には、ワークフロー案件処理API（スクリプト開発モデル）のアプリケーションデータ（入力アイテムデータ）のオブジェクトをJSONにフォーマットして指定します。

- 日付型・タイムスタンプ型
  - ISO 8601 の規格に従った文字列（YYYYMMDDThhmmss+0900）にて指定します。
- （例）JSON

```
"formaUserParam": {
   "items": {
      "textbox1": "あいうえお",
      "number1": 1000,
      "calendar1": "2010-07-28T22:25:51Z"
   }
}
```

アプリケーションデータ（テーブルアイテムデータ）

基本的には、ワークフロー案件処理API（スクリプト開発モデル）のアプリケーションデータ（テーブルアイテムデータ）のオブジェクトをJSONにフォーマットして指定します。

- 日付型・タイムスタンプ型
  - ISO 8601 の規格に従った文字列（YYYYMMDDThhmmss+0900）にて指定します。
- （例）JSON

```
"formaUserParam": {
   "items": {
      "tb1": [
         {
            "tb1_textbox1": "明細1 - りんご",
            "tb1_textbox2": "明細1 - バナナ"
         },
         {
            "tb1_textbox1": "明細1 - ぶどう",
            "tb1_textbox2": "明細1 - 苺"
         }
      ],
      "tb2": [
         {
            "tb2_number1": 345.678,
            "tb2_calendar1": "2010-07-28T22:25:51Z"
         },
         {
            "tb2_number1": 99999,
            "tb2_calendar1": "2015-07-23T22:25:51Z"
         }
      ]
   }
}
```

アプリケーションデータ（ファイルアップロードアイテムデータ）

ファイルアップロードアイテムデータについてはトークン方式にて指定を行います。

1. ファイルアップロード用のWeb APIを実行します。
   リクエスト情報は、以下のように送信します。

リクエストヘッダ

フィールド名	値
Content-Type	multipart/form-data
Accept	application/json

リクエストパラメータ

フィールド名	値
file	ファイルデータ

コラム

ファイルのアップロードに成功すると、トークンとファイル名が返却されます。
複数のファイルをアップロードする場合は、複数回リクエストを発行し、トークンを取得してください。

2. 取得したトークンをアプリケーションデータ（ファイルアップロードアイテムデータ）のJSONに指定します。
   - （例）JSON

```
"formaUserParam": {
   "items": {
     ...
   },
   "files": [
            {
               "token": "8e9vzec9jacb1cy",
               "fileName": "sample_file.txt",
               "uploadItemId": "attach_fileupload_item1",
               "notes": "サンプル備考１"
            },
            {
               "token": "8e9vzikscacfwcy",
               "fileName": "sample_file.jpg",
               "uploadItemId": "attach_fileupload_item1",
               "notes": "サンプル備考２"
            }
         ]
}
```

注意事項

注意

- Web APIはワークフロー案件処理Web APIを拡張した機能であり、ワークフロー案件処理Web APIの同様の制限が存在します。
- Web APIのContent-Type・Acceptは、application/jsonのみの対応となり、application/xmlには対応していません。
- formaUserParamの文字列型のパラメータに対して、ISO 8601 の規格に従った日付文字列が送信された場合は、日付型・タイムスタンプ型として処理されます。
- Web APIを利用してワークフローの申請を行う場合、パラメータ（アプリケーションデータ）に指定できるアイテムは、申請ノードに設定されている画面遷移のフォームに含まれるものに限定されます。
  Web APIを利用してワークフローの承認を行う場合、パラメータ（アプリケーションデータ）に指定できるアイテムは、承認ノードに設定されている画面遷移のフォームに含まれるものに限定されます。
- 以下のWeb APIを実行した場合は、現状、formaUserParamパラメータを指定してもアプリケーションデータは更新されません。

IM-FormaDesigner

メソッド	パス	説明
POST	/api/forma/imw/deny	否認処理を実行します。
POST	/api/forma/imw/discontinue	取止め処理を実行します。
POST	/api/forma/imw/reserve	保留処理を実行します。
POST	/api/forma/imw/reserveCancel	保留解除処理を実行します。
POST	/api/forma/imw/pullBack	引戻し処理を実行します。
POST	/api/forma/imw/sendBack	差戻し処理を実行します。

IM-BIS

メソッド	パス	説明
POST	/api/bis/imw/deny	否認処理を実行します。
POST	/api/bis/imw/discontinue	取止め処理を実行します。
POST	/api/bis/imw/reserve	保留処理を実行します。
POST	/api/bis/imw/reserveCancel	保留解除処理を実行します。
POST	/api/bis/imw/pullBack	引戻し処理を実行します。
POST	/api/bis/imw/sendBack	差戻し処理を実行します。

認可設定を行う

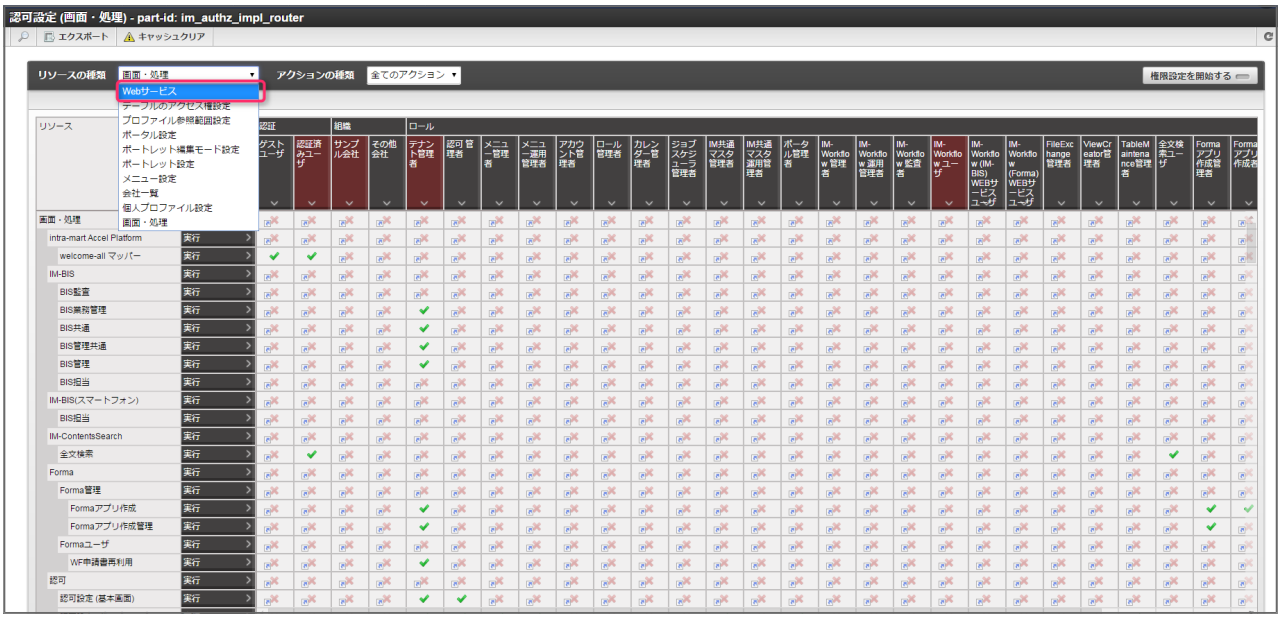
ワークフロー案件処理Web APIでは認可チェックによる権限制御を採用しているため、Web APIを実行するユーザは認可チェックの対象に含まれます。
実行権限を付与するには、以下の2つの方法があります。

- 認可ポリシーを設定する。
- ロール設定を行う。

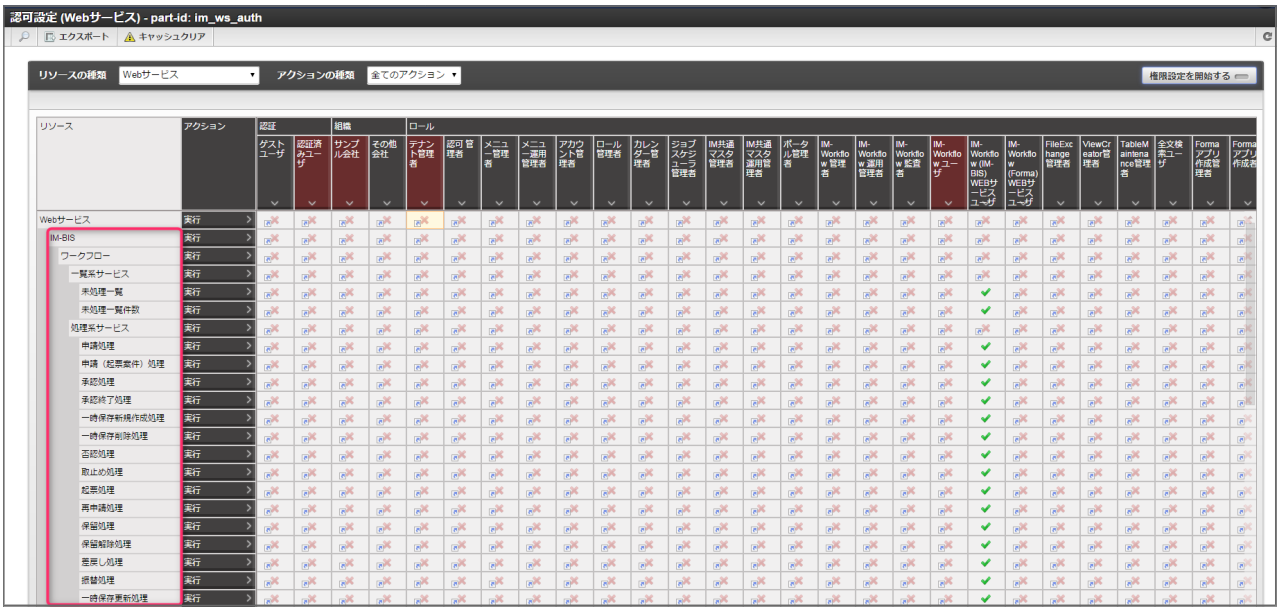
認可ポリシーを設定する

Web APIの認可リソースに対して、認可設定を行います。
個別のWeb APIごとに権限設定を行いたい場合は、この認可設定を利用します。

1. テナント管理者としてログインし、認可設定画面を表示します。
   - 「グローバルナビ」-「テナント管理」-「認可」をクリックし、認可設定画面を表示します。
2. リソースの種類を「Webサービス」に変更します。
   - リソースの種類を選択することで、Webサービスの認可リソースに対する設定画面が表示されます。



3. Web APIの認可リソースに対してポリシーを作成します。
   - 実行権限を付与したいリソースに対するポリシーを作成します。
     - リソースは、Web APIのURLごとに用意されています。



ロール設定を行う

Web APIへの実行権限があらかじめ付与された状態のロールが提供されています。
Web APIの実行ユーザをロールに所属されることで、実行権限を付与することが可能です。

- 「IM-Workflow (Forma) WEBサービス ユーザ」ロール
  - IM-FormaDesigner ワークフロー案件処理Web APIの認可リソース全てへの実行権限が付与されています。
- 「IM-Workflow (IM-BIS) WEBサービス ユーザ」ロール
  - IM-BIS ワークフロー案件処理Web APIの認可リソース全てへの実行権限が付与されています。

認証設定を行う

Web APIを実行するには、intra-mart Accel Platform の認証を通過する必要があります。
ここでは、以下3つの認証方法について説明します。

- Basic認証
- OAuth認証
- Cookie認証

> **注意**
>
> 認証ユーザと案件の処理を実行するユーザは、それぞれ異なるアカウントが指定可能です。

Basic認証

Web APIを実行するユーザのアカウント情報を利用し、Basic認証の形式でエンコーディングした情報をHTTPリクエストヘッダーで送信します。
全ての案件処理を管理者ユーザのアカウントで行う場合には、この認証方法を利用します。

以下は、リクエスト情報の例です。

1. Basic認証の形式（アカウントのユーザコード:アカウントのパスワード）の文字列を作成します。
   - （例）ueda:ueda
2. Base64でエンコードします。
   - （例）YW95YWdpOmFveWFnaQ==
3. エンコーディングした文字列をリクエストヘッダーとして送信します。
   - （例）Authorization: Basic YW95YWdpOmFveWFnaQ==

> **注意**
>
> Basic認証を利用する場合は、Web APIのURLの前に/basicを付与する必要があります。
>
> - （例）/basic/api/forma/imw/apply

> **注意**
>
> Basic認証を利用する場合は、盗聴や改竄の危険性があるため、SSLを有効にしてください。

OAuth認証

intra-mart Accel Platform で提供されているOAuth2の認証フローに従いアクセストークンを取得し、HTTPリクエストヘッダーで送信します。

1. OAuth認証を利用するためのクライアントアプリケーションを intra-mart Accel Platform に登録します。
   - クライアントアプリケーションの登録方法については、「 OAuth 管理者操作ガイド 」の「 クライアントアプリケーションの登録 」を参照してください。
2. OAuth2の認証フローに従い、アクセストークンを取得します。
   - アクセストークンの取得方法については、「 OAuth プログラミングガイド 」の「 クライアントアプリケーションからOAuth認証機能を利用する方法 」を参照してください。
3. アクセストークンをリクエストヘッダーとして送信します。
   - （例）Authorization: Bearer 718fedeab471477fa1c0deef626e0b0d

> **注意**
>
> OAuth認証を利用する場合は、Web APIのURLの前に/oauthを付与する必要があります。
>
> - （例）/oauth/api/forma/imw/apply

> **注意**
>
> OAuth認証を利用する場合には、下記のモジュールを構成に含める必要があります。
>
> - IM-FormaDesigner ワークフロー案件処理Web APIにてOAuth認証を利用する場合
>   - IM-FormaDesigner OAuth認証モジュール
> - IM-BIS ワークフロー案件処理Web APIにてOAuth認証を利用する場合
>   - IM-BIS OAuth認証モジュール

Cookie認証

Cookieに含まれるセッションIDを利用して認証を行います。
特別な認証処理を行わず、Cookieに紐づくセッションの認証状態でアクセスします。

1. リクエストヘッダーを送信します。

- （例） Cookie: JSESSIONID=aaa8M4I9SgzLK7z0nbe7u

認証状態の維持について

Web APIを実行する際に、後続のリクエスト時に認証セッションを維持するかどうかを制御するためのHTTPリクエストヘッダ X-Intramart-Session が提供されています。
詳細については、「Web API Maker プログラミングガイド」の「セッション管理」を参照してください。

HTTPリクエストヘッダ X-Intramart-Session

keep	認証状態（セッション）を維持します。API 実行後もログアウトしません（デフォルト）。
once	API 実行前に未認証だった場合のみ、API 実行後にログアウトします。
never	認証状態（セッション）を破棄して、常にログアウトします。

クライアントアプリケーションを実装する

ここでは、ワークフロー案件処理 Web APIを利用して実際にファイルアップロード・申請・承認を行うサンプルのリクエスト・レスポンス情報を紹介します。
認証方法については、Basic認証を利用します。

ファイルアップロード Web APIの実行

リクエスト情報

- リクエストURI
  - basic/api/forma/imw/fileupload
- リクエストメソッド
  - POST
- リクエストヘッダ
  - Content-Type : multipart/form-data
  - Accept : application/json
  - Authorization : Basic dWVkYTp1ZWRh
  - X-Intramart-Session : once
- リクエストパラメータ
  - file : ファイルデータ

レスポンス情報

- レスポンスコード
  - 200
- レスポンスヘッダ
  - Pragma : no-cache
  - Date : Sun, 26 Jul 2015 23:51:00 GMT
  - Cache-Control : no-store
  - Server : Resin/4.0.44
  - Content-Length : 81
  - Content-Type : application/json; charset=UTF-8
- レスポンスボディ

```
{
 'error': false,
 'data': {
   'token': '5ieu48642uaowzo',
   'fileName': 'fileupload (2).js'
 }
}
```

申請処理 Web APIの実行

リクエスト情報

- リクエストURI
  - basic/api/forma/imw/apply
- リクエストメソッド
  - POST
- リクエストヘッダ
  - Content-Type : application/json
  - Accept : application/json
  - Authorization : Basic dWVkYTp1ZWRh
  - X-Intramart-Session : once
- リクエストボディ

```
{
 "applyBaseDate": "2015/07/28",
 "matterName": "sample_matter",
 "applyExecuteUserCode": "aoyagi",
 "userParam": {},
 "formaUserParam": {
   "items" : {
     "textbox1" : "あいうえお"
   },
   "files": [
           {
              "token": "5ieu48a0fpe2hzo",
              "fileName": "サンプルファイル名１",
              "uploadItemId": "attach_fileupload_item1",
              "notes": "サンプル備考１"
           }
  ]
 },
 "applyAuthUserCode": "aoyagi",
 "flowId": "web_api_test"
}
```

レスポンス情報

- レスポンスコード
  - 200
- レスポンスヘッダ
  - Pragma : no-cache
  - Date : Mon, 27 Jul 2015 00:22:12 GMT
  - Cache-Control : no-store
  - Server : Resin/4.0.44
  - Content-Length : 121
  - Content-Type : application/json; charset=UTF-8
- レスポンスボディ

```
{
  error: false
  data: {
    systemMatterId: 'ma_5ieu48aciukd9zo'
    userDataId: '5ieu48acirfd6zo'
    matterNumber: '0000000011'
  }
}
```

承認処理 Web APIの実行

リクエスト情報

- リクエストURI
  - basic/api/forma/imw/approve
- リクエストメソッド
  - POST

- リクエストヘッダ
  - Content-Type : application/json
  - Accept : application/json
  - Authorization : Basic dWVkYTp1ZWRh
  - X-Intramart-Session : once
- リクエストボディ

```
{
 "executeUserCode": "aoyagi",
 "userParam": {},
 "formaUserParam": {
   "items" : {
     "textbox1" : "あいうえお"
   }
 },
 "authUserCode": "aoyagi",
 "systemMatterId": "ma_5ieu48aciukd9zo"
}
```

レスポンス情報

- レスポンスコード
  - 200
- レスポンスヘッダ
  - Pragma : no-cache
  - Date : Mon, 27 Jul 2015 00:34:17 GMT
  - Cache-Control : no-store
  - Server : Resin/4.0.44
  - Content-Length : 15
  - Content-Type : application/json; charset=UTF-8
- レスポンスボディ

```
{
  error: false
}
```

ファイルアップロードアイテム API

APIの位置づけ

IM-FormaDesigner for Accel Platform では、ユーザによる画面操作を経由することなく、ユーザプログラム上からファイルアップロードアイテムの操作を行うための API を提供しています。

API を利用することで、以下のようなことが実現できます。

- ユーザプログラム内で、配置されている画面アイテム「ファイルアップロード」へ、ファイルの登録、削除、添付されているファイル情報の取得を行う

API のインタフェースの詳細については「IM-FormaDesigner for Accel Platform APIドキュメント」を参照してください

本ドキュメントでは、API の基本的な利用方法と注意事項について説明します。

JavaEE開発モデル

ここでは、ファイルアップロードアイテムAPI（JavaEE開発モデル）の基本的な利用方法を説明します。
APIのインタフェース情報については「IM-FormaDesigner for Accel Platform APIドキュメント（JavaEE開発モデル）」を参照してください。

> **注意**
> この章では、IM-FormaDesigner の API を利用した場合のサンプルコードを掲載しています。

コンストラクタ

- コンストラクタ
  - FormaFileUploadItemManager(String applicationId)
  - パラメータ
    applicationId アプリケーションID

ファイル登録処理の実装

> **注意**
> 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完了した時点で、ファイル登録が確定します。
> データ操作完了までは、一時保存の状態です。

パラメータ

ファイル登録処理APIを実行するには、ファイル情報、ユーザデータID、およびプロセスキーの3つのパラメータが必要です。

- パラメータ（ファイル情報）

  登録するファイルの情報です。下記は必須項目です。

パラメータ（ファイル情報）

パラメータ	説明
ファイル	登録するファイルの実体を、InputStreamで指定します。
ファイル名	登録するファイル名を指定します。
アイテム識別ID	登録する画面アイテム「ファイルアップロード」の アイテム識別IDを指定します。

```
// パラメータ（ファイル情報）の作成
final List<FormaFileUpload> fileUploadItems = new ArrayList<FormaFileUpload>();
final FormaFileUpload fileUploadItem = new FormaFileUpload();
final PublicStorage ps = new PublicStorage("sample/sampleFile.txt");
fileUploadItem.setFile(ps.open());
fileUploadItem.setFileName("sampleFile.txt");
fileUploadItem.setUploadItemId("attach_fileupload_item1");
fileUploadItems.add(fileUploadItem);
```

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。
  この項目は、必須項目です。

ファイル登録処理を実装する

例として、後処理プログラム内でファイル登録処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのファイル登録処理メソッドにパラメータを指定して、実行します。

```
public void regist(final PostProcessParameter formaParam, final Map<String, Object> sendParam) throws Exception {
  final FormaFileUploadItemManager manager = new FormaFileUploadItemManager(formaParam.getApplicationId());

  // パラメータ（ファイル情報）の作成
  final List<FormaFileUpload> fileUploadItems = new ArrayList<FormaFileUpload>();
  final FormaFileUpload fileUploadItem = new FormaFileUpload();
  final PublicStorage ps = new PublicStorage("sample/sampleFile.txt");
  fileUploadItem.setFile(ps.open());
  fileUploadItem.setFileName("sampleFile.txt");
  fileUploadItem.setUploadItemId("attach_fileupload_item1");
  fileUploadItems.add(fileUploadItem);

  // 登録処理の実行
  manager.registerFileTemp(fileUploadItems, formaParam.getInsertId(), formaParam.getProcessKey());
}
```

戻り値

戻り値はありません。

- エラー時は jp.co.intra_mart.foundation.forma.exception.FormaSystemException クラス、または
  jp.co.intra_mart.foundation.forma.exception.FormaApplicationException クラスがスローされます。

ファイル削除処理の実装

> **注意**
>
> 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完了した時点で、ファイル削除が確定します。
> データ操作完了までは、一時保存の状態です。

パラメータ

ファイル削除処理を実行するには、ユーザデータID、アイテム識別ID、ファイルID、およびプロセスキーの4つのパラメータが必要です。

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（アイテム識別ID）

  アップロードしたファイルを画面アイテムと関連付けるためのIDです。
  この項目は、必須項目です。

- パラメータ（ファイルID）

  登録したファイルを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。
  この項目は、必須項目です。

ファイル削除処理を実装する

例として、後処理プログラム内でファイル削除処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのファイル削除処理メソッドにパラメータを指定して、実行します。

```
public void regist(final PostProcessParameter formaParam, final Map<String, Object> sendParam) throws Exception {
   final FormaFileUploadItemManager manager = new FormaFileUploadItemManager(formaParam.getApplicationId());
   boolean resultFlg = manager.deleteFile(formaParam.getInsertId(), "attach_fileupload_item1", "fileId", formaParam.getProcessKey());
}
```

> **注意**
>
> ファイルIDは、*ファイル情報取得処理の実装*から取得することが可能です。

戻り値

boolean 型のファイル削除結果が、戻り値として返却されます。以下の情報が取得できます。

戻り値

パラメータ	説明
ファイル削除結果	正しく削除された場合はtrue、そうでない場合はfalseが返却されます。

- エラー時は jp.co.intra_mart.foundation.forma.exception.FormaSystemException クラスがスローされます。

ファイル情報取得処理の実装

パラメータ

ファイル情報取得処理を実行するには、ユーザデータIDが必要です。
プロセスキーを指定した場合、登録済みのファイル情報とセッションに保存されている一時保存状態のファイル情報を取得できます。
プロセスキーを指定していない場合、登録済みのファイル情報以外は取得できません。

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。

ファイル情報取得処理を実装する

例として、後処理プログラム内でファイル情報取得処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのファイル情報取得処理メソッドにパラメータを指定して、実行します。

```
public void regist(final PostProcessParameter formaParam, final Map<String, Object> sendParam) throws Exception {
    final FormaFileUploadItemManager manager = new FormaFileUploadItemManager(formaParam.getApplicationId());
    Map<String, List<FileUploadFileInfoModel>> result = manager.getFilesInfo(formaParam.getInsertId(), formaParam.getProcessKey());
}
```

戻り値

ファイル情報マップ が、戻り値として返却されます。
以下の情報が取得できます。
ファイルの実体を取得する場合は、 *ファイル取得処理の実装*を利用してください。

ファイル情報マップ

フィールド識別ID (String)	ファイル情報リスト(List<FileUploadFileInfoModel>)	
	fileId (String)	ファイルID
	fileName (String)	ファイル名
	notes (String)	備考
	uploadItemId (String)	アイテム識別ID
	recordDate (Date)	最終更新日
	recordUserCd (String)	更新ユーザコード
	createDate (Date)	作成日
	createUserCd (String)	登録ユーザコード
	status (String)	ステータス

ファイル取得処理の実装

パラメータ

ファイル取得処理を実行するには、ユーザデータID、ファイルIDの2つのパラメータが必要です。
プロセスキーを指定した場合、登録済みのファイル情報とセッションに保存されている一時保存状態のファイル情報を取得できます。
プロセスキーを指定していない場合は、登録済みのファイル情報以外は取得できません。

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（ファイルID）

  登録したファイルを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。

ファイル情報取得処理を実装する

例として、後処理プログラム内でファイル情報取得処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。

作成したファイルアップロードアイテムAPIのファイル情報取得処理メソッドにパラメータを指定して、実行します。

```
public void regist(final PostProcessParameter formaParam, final Map<String, Object> sendParam) throws Exception {
    final FormaFileUploadItemManager manager = new FormaFileUploadItemManager(formaParam.getApplicationId());
    FileUploadFileInfoModel result = manager.getFile(formaParam.getInsertId(), "fileId", formaParam.getProcessKey());
}
```

> **注意**
>
> ファイルIDは、*ファイル情報取得処理の実装*から取得することが可能です。

> **注意**
>
> プロセスキーは「*前処理*」、「*後処理*」、「*入力チェックプログラム*」のパラメータからのみ
> 取得することが可能です。

戻り値

ファイル情報マップ が、戻り値として返却されます。以下の情報が取得できます。

ファイル情報マップ(FileUploadFileInfoModel)	
file (InputStream)	ファイルデータ
fileId (String)	ファイルID
fileName (String)	ファイル名
notes (String)	備考
uploadItemId (String)	アイテム識別ID
recordDate (Date)	最終更新日
recordUserCd (String)	更新ユーザコード
createDate (Date)	作成日
createUserCd (String)	登録ユーザコード
status (String)	ステータス

スクリプト開発モデル

ここでは、ファイルアップロードアイテムAPI（スクリプト開発モデル）の基本的な利用方法を説明します。
APIのインタフェース情報については「IM-FormaDesigner for Accel Platform APIドキュメント（スクリプト開発モデル）」を参照してください。

> **注意**
>
> この章では、IM-FormaDesigner の API を利用した場合のサンプルコードを掲載しています。

コンストラクタ

- コンストラクタ
  - FormaFileUploadItemManager(applicationId)
  - パラメータ
    applicationId アプリケーションID

ファイル登録処理の実装

> **注意**
>
> 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完了した時点で、ファイル登録が確定します。
> データ操作完了までは、一時保存の状態です。

パラメータ

ファイル登録処理APIを実行するには、ファイル情報、ユーザデータID、およびプロセスキーの3つのパラメータが必要です。

- パラメータ（ファイル情報）

  登録するファイルの情報です。下記は必須項目です。

パラメータ（ファイル情報）

パラメータ	説明
ファイルパス	ファイルパスを指定します。 パブリックストレージにあるファイルを指定することが可能です。
アイテム識別ID	登録する画面アイテム「ファイルアップロード」のアイテム識別IDを指定します。

```
// パラメータ（ファイル情報）の作成
var files = [];
var fileObj = {};
fileObj.uploadItemId = 'attach_fileupload_item1';
fileObj.filePath = 'pdftest.pdf';
files.push(fileObj);
```

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。
  この項目は、必須項目です。

ファイル登録処理を実装する

例として、後処理プログラム内でファイル登録処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのファイル登録処理メソッドにパラメータを指定して、実行します。

```
function regist(formaParam, sendParam) {
   var manager = new FormaFileUploadItemManager(formaParam.applicationId);

   // パラメータ（ファイル情報）の作成
   var files = [];
   var fileObj = {};
   fileObj.uploadItemId = 'attach_fileupload_item1';
   fileObj.filePath = 'pdftest.pdf';
   files.push(fileObj);

   // 登録処理の実行
   var result = manager.registerFileTemp(files, formaParam.insertId, formaParam.processKey);
}
```

戻り値

プロパティ	詳細
error (Boolean)	エラーフラグ 処理中にエラーが発生した場合はtrue、 ない場合はfalseを設定します。
errorMessage (String)	エラーメッセージ

ファイル削除処理の実装

> **注意**
>
> 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完了した時点で、ファイル削除が確定します。
> データ操作完了までは、一時保存の状態です。

パラメータ

ファイル削除処理を実行するには、ユーザデータID、アイテム識別ID、ファイルID、およびプロセスキーの4つのパラメータが必要です。

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（アイテム識別ID）

  アップロードしたファイルを画面アイテムと関連付けるためのIDです。
  この項目は、必須項目です。

- パラメータ（ファイルID）

  登録したファイルを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。
  この項目は、必須項目です。

ファイル削除処理を実装する

例として、後処理プログラム内でファイル削除処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのファイル削除処理メソッドにパラメータを指定して、実行します。

```
function regist(formaParam, sendParam) {
    var manager = new FormaFileUploadItemManager(formaParam.applicationId);
    var result = manager.deleteFile(formaParam.insertId, "attach_fileupload_item1", "fileId", formaParam.processKey);
}
```

> **注意**
>
> ファイルIDは、*ファイル情報取得処理の実装*から取得することが可能です。

戻り値

プロパティ	詳細
error (Boolean)	エラーフラグ 処理中にエラーが発生した場合はtrue、 ない場合はfalseを設定します。
errorMessage (String)	エラーメッセージ

ファイル情報取得処理の実装

パラメータ

ファイル情報取得処理を実行するには、ユーザデータIDが必要です。
プロセスキーを指定した場合、登録済みのファイル情報とセッションに保存されている一時保存状態のファイル情報を取得できます。
プロセスキーを指定していない場合、登録済みのファイル情報以外は取得できません。

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。

ファイル情報取得処理を実装する

例として、後処理プログラム内でファイル情報取得処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのフファイル情報取得処理メソッドにパラメータを指定して、実行します。

```
function regist(formaParam, sendParam) {
    var manager = new FormaFileUploadItemManager(formaParam.applicationId);
    var result = manager.getFilesInfo(formaParam.insertId, formaParam.processKey);
}
```

戻り値

ファイル情報オブジェクト が、戻り値として返却されます。
以下の情報が取得できます。
ファイルの実体を取得する場合は、*ファイル取得処理の実装*を利用してください。

プロパティ	詳細		
error (Boolean)	エラーフラグ 処理中にエラーが発生した場合はtrue、ない場合はfalseを設定します。		
errorMessage (String)	エラーメッセージ		
data (Object)	フィールド識別ID (String)	ファイル情報リスト(Array)	
		file_id (String)	ファイルID

プロパティ詳細		
	file_name (String)	ファイル名
	notes (String)	備考
	record_date (Date)	最終更新日
	record_user_cd (String)	更新ユーザコード
	create_date (Date)	作成日
	create_user_cd (String)	登録ユーザコード
	status (String)	ステータス

ファイル取得処理の実装

パラメータ

ファイル取得処理を実行するには、ユーザデータID、ファイルIDの2つのパラメータが必要です。
プロセスキーを指定した場合、登録済みのファイル情報とセッションに保存されている一時保存状態のファイル情報を取得できます。
プロセスキーを指定していない場合は、登録済みのファイル情報以外は取得できません。

- パラメータ（ユーザデータID）

  作成したアプリケーションデータを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（ファイルID）

  登録したファイルを一意に識別できるID情報です。
  この項目は、必須項目です。

- パラメータ（プロセスキー）

  プロセスキーです。

ファイル情報取得処理を実装する

例として、後処理プログラム内でファイル情報取得処理に必要なパラメータを作成します。
アプリケーションIDを指定して、ファイルアップロードアイテムAPIのインスタンスを作成します。
作成したファイルアップロードアイテムAPIのファイル情報取得処理メソッドにパラメータを指定して、実行します。

```
function regist(formaParam, sendParam) {
   var manager = new FormaFileUploadItemManager(formaParam.applicationId);
   var  result = manager.getFile(formaParam.insertId, 'fileId', formaParam.processKey);
}
```

> **注意**
>
> ファイルIDは、*ファイル情報取得処理の実装*から取得することが可能です。

> **注意**
>
> プロセスキーは「*前処理*」、「*後処理*」、「*入力チェックプログラム*」のパラメータからのみ
> 取得することが可能です。

戻り値

ファイル情報オブジェクト が、戻り値として返却されます。以下の情報が取得できます。

プロパティ詳細		
error (Boolean)	エラーフラグ 処理中にエラーが発生した場合はtrue、 ない場合はfalseを設定します。	
errorMessage (String)	エラーメッセージ	
data (Object)	data (InputStream)	ファイルデータ
	file_id (String)	ファイルID
	file_name (String)	ファイル名
	notes (String)	備考
	record_date (Date)	最終更新日
	record_user_cd (String)	更新ユーザコード
	create_date (Date)	作成日
	create_user_cd (String)	登録ユーザコード
	status (String)	ステータス

注意事項

ファイルアップロードアイテムAPIの対応表

ファイルアップロードアイテムAPIが持っている処理によって、対応している機能が異なります。
ファイルアップロードアイテムAPIが対応している機能については、以下のとおりです。

機能	ファイル登録処理	ファイル削除処理	ファイル情報取得処理	ファイル取得処理
前処理プログラム	⛔	⛔	✔	✔
後処理プログラム	✔	✔	✔	✔
入力チェックプログラム	⛔	⛔	✔	✔
ユーザプログラム (IM-Workflow)	⛔	⛔	✔	✔
実行中の外部連携	⛔	⛔	✔	✔
バッチなど 外部からの処理	⛔	⛔	✔	✔

> **注意**
>
> 前処理プログラムでファイル登録、削除を行いたい場合は、ファイルアップロードアイテムAPIではなく、「 *前処理* 」の返却値を利用してください。

IM-Workflow で、申請、承認前に任意のプログラムを実行する方法

ここでは、IM-FormaDesigner のアプリケーション種別「 IM-Workflow 」にて、申請、承認ボタンを押下したときに、任意のプログラムを実行する方法を説明します。
IM-Workflow の標準処理画面を表示する前のタイミングで任意の処理を実行し、処理がエラーの場合は、画面上にメッセージを表示します。
これにより、アクション設定とは異なるタイミングで任意の処理を実行することができます。

Contents

コラム

アクション設定は、 IM-BIS 導入環境で利用することができます。

コラム

IM-BIS で作成したBISフロー／ワークフローでも同様の手順にて任意のプログラムを実行できます。

概要

任意のプログラムを下記の方法に沿って作成し、テンプレートHTMLに組み込みます。
テンプレートHTMLは、テナント上のすべてのアプリケーションで共有されますが、共有範囲を限定することも可能です。
詳細は、「 intra-mart Developer Blog - チュートリアル 」を参照してください。

任意プログラムの実装方法

任意のプログラムを実行するためには、以下を実装する必要があります。

- 配列 window.forma.optionalProcess への実行したい関数の追加
- 戻り値の設定 下記のプロパティを持つObject型のオブジェクトを作成
  - 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
  - message : エラーメッセージ　　設定したメッセージは実行画面に出力されます。

実装例

```
if (!window.forma) {
    window.forma = {};
}
if (!window.forma.optionalProcess) {
    window.forma.optionalProcess = [];
}


var imSampleExecutor = function(request) {
    var csjsResult = {
        error: false,
        message: ''
    };

    /*
     * ここに実行する処理を記述します。
     */

    //処理がエラーの場合、実行画面にメッセージを返却します。
    csjsResult.error = false;
    csjsResult.message = '';
    return csjsResult;
}

//window.forma.optionalProcessは、申請/承認ボタン押下時に実行されます。
window.forma.optionalProcess.push(imSampleExecutor);
```

テンプレートHTMLの実装

テンプレートHTMLに実行したいプログラムを読み込みます。

IM-FormaDesigner のアプリケーションが実行される画面は、「テンプレートHTML」というベースとなるHTMLファイル上に各画面アイテムのHTMLを配置して生成する仕組みです。
scriptタグを記述することで、アプリケーションの実行画面にて任意のスクリプトファイルを読み込ませることができます。
詳細は「 intra-mart Developer Blog - テンプレートHTMLによるスクリプトの外出し・共通化 」を参照してください。

```
<imart type="head">
<link rel="stylesheet" type="text/css" href="forma/css/print.css" media="print"></link>
<link rel="stylesheet" type="text/css" href="forma/css/style.css"></link>
<link rel="stylesheet" type="text/css" href="forma/css/forma-ui-icons.css"></link>
<imart type="condition" validity=imfrBindData.isSmartphone>
<link rel="stylesheet" type="text/css" href="forma/css/smartphone.css"></link>
</imart>
<link rel="stylesheet" type="text/css" href="annotation/css/annotation_style.css"></link>
<link rel="stylesheet" type="text/css" href="annotation/css/annotation-ui-icons.css"></link>

<imart type="jsspRpc" name="formaServerLogic" page="forma/common/ajax/server_logic" />
<script type="text/javascript" src="forma/csjs/im_forma_click.js"></script>
<script type="text/javascript" src="forma/csjs/im_forma.js"></script>

%HEADER%

<imart type="string" value=imfrBindData.outputScript></imart>
<imart type="imACMSearch" name="imMasterSaerchObj"/>
<script type="text/javascript" src="csjs/im_window.js"></script>
<script type="text/javascript" src="csjs/im_json.js"></script>
<script type="text/javascript" src="csjs/im_date.js"></script>
<script type="text/javascript" src="csjs/im_code_point_util.js"></script>
<script type="text/javascript" src="forma/csjs/im_forma_util.js"></script>
<script type="text/javascript" src="forma/csjs/im_forma_tabs.js"></script>
<!-- 任意のプログラムを追加。 -->
<script type="text/javascript" src="forma/csjs/test.js"></script>
<script type="text/javascript" src="forma/csjs/jquery.contextmenu.r2.packed.js"></script>

<imart type="condition" validity=imfrBindData.isFooterDisplay negative>
```

コラム

外部ファイルで定義したスクリプトをアプリケーションに組み込む手順については、「 intra-mart Developer Blog - チュートリアル 」を参照してください。