目次

改訂情報

変更年月日	変更内容
2016-10-01	初版

はじめに

本書の目的

本書は、IM-BPM for Accel Platform（以下 IM-BPM）におけるそれぞれの機能を拡張する仕組の詳細および、基本的な使用方法も併せて説明します。

説明範囲は以下のとおりです。

- プロセス定義で使用する内部・外部モジュールについて
- IM-BPMのサービスを使用してのプロセスの操作方法
- 他アプリケーションとの連携方法

対象読者

本書では以下のユーザを対象としています。

- IM-BPMを利用して処理を実装したい
- IM-BPMと連携した機能を実装したい

なお、本書では次の内容を理解していることが必須となります。

- IM-BPMを理解している
- IM-BPM Designerの基本機能を理解している
- intra-mart Accel Platformを理解している

サンプルコードについて

本書に掲載されているサンプルコードは可読性を重視しており、性能面や保守性といった観点において必ずしも適切な実装ではありません。
開発においてサンプルコードを参考にされる場合には、上記について十分に注意してください。

本書の構成

本書は次の構成となっています。

- *プロセス定義*

  プロセス定義における内部・外部モジュールについて説明します。
  スクリプトタスク、サービスタスクのプログラミング方法やあ設定方法を説明します。
  リスナの用途やプログラミング方法について説明します。

- *サービスを使用してのプロセスの操作方法*

  プロセスインスタンス、タスクの操作方法を説明します。
  その他、メッセージやシグナルについても操作方法も説明します。

- *他アプリケーションとの連携方法*

intra-mart Accel Platformのアプリケーションとの連携について説明します。

プロセス定義

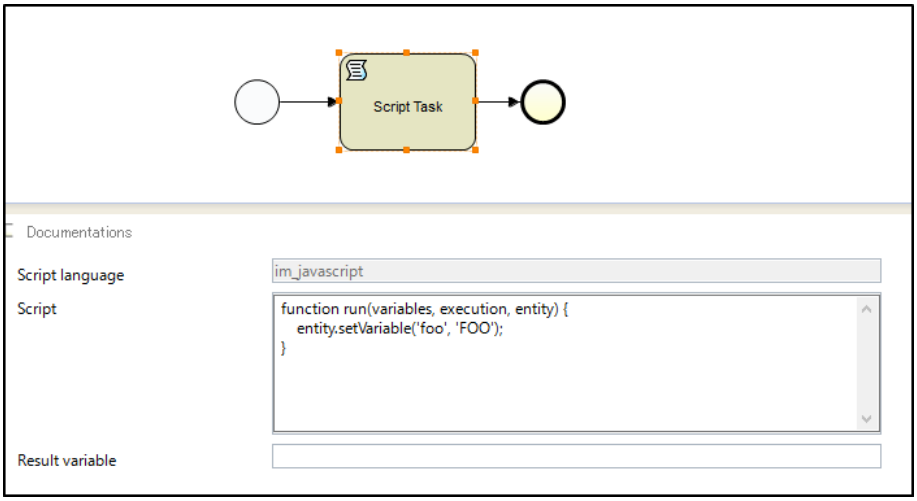
ここでは IM-BPMのプロセス定義における各機能の作成方法について説明します。

スクリプトタスク

項目
- スクリプトを作成する
- 結果を返却する

スクリプトを作成する

スクリプトタスクのスクリプトは IM-BPM Designerでスクリプトタスク作成時に下記のコードが自動生成されます。
スクリプト言語のエンジンは、スクリプト開発モデルのスクリプトエンジンを使用しています。



図：スクリプトタスク

```
function run(variables, execution, entity) {
  entity.setVariable('foo', 'FOO');
}
```

引数のオブジェクトの説明をします。

- variables
  変数です。変数名をキーとしているオブジェクトです。
  (例)param1という変数を取得する場合。var param1 = variables.param1;
- execution
  実行状態で保持しているオブジェクトです。取得できる情報は、以下になります。
  id : 実行状態のID（string）
  processInstanceId : プロセスインスタンスID（string）
  processDefinitionId : プロセス定義ID（string）
  businessKey : 業務キー（string）
  activityId : アクティビティID（string）
  isActive : アクティブか（boolean）
  isConcurrent : 同期か（boolean）
  isScope : スコープか（boolean）

isEventScope : イベントスコープか（boolean）
parentId : 親のID（string）
name : 名前（string）
lockTime : ロックタイム（date）
superExecution : 親のID（string）
forcedUpdate : 変数の強制更新楽観ロック（boolean）
suspensionState : 無効状態（int）0 : 有効、1 : 無効
cachedEntityState : エンティティ状態（int）
(例) プロセス定義IDを取得する場合。var processDefinitionId = execution.processDefinitionId;

- entity
  実行状態のエンティティです。`jp.co.intra_mart.activiti.engine.impl.persistence.entity.ExecutionEntity` クラスです。
  http://www.intra-mart.jp/apidoc/bpm/apilist-bpm-javadoc/doc/index.html

結果を返却する

スクリプトタスクの結果変数名に設定したキーで、スクリプトの戻り値がプロセスインスタンスの変数として設定されます。
結果変数名を設定しない場合は、戻り値を返却してもプロセスインスタンスの変数として設定されることはありません。

```
function run(variables, execution, entity) {
  var a = variables.a;
  var b = variables.b;
  return {'ab' : a + b};
}
```

複数の値を返却することも可能です。

```
function run(variables, execution, entity) {
  var a = variables.a;
  var b = variables.b;
  var c = variables.b;
  return {'ab' : a + b, 'abc' : a + b + c};
}
```

> **コラム**
>
> 上記返却の場合、%結果変数名%のMapオブジェクトが変数として設定され、そのMapのキーに”ab”と”abc”が設定されます。

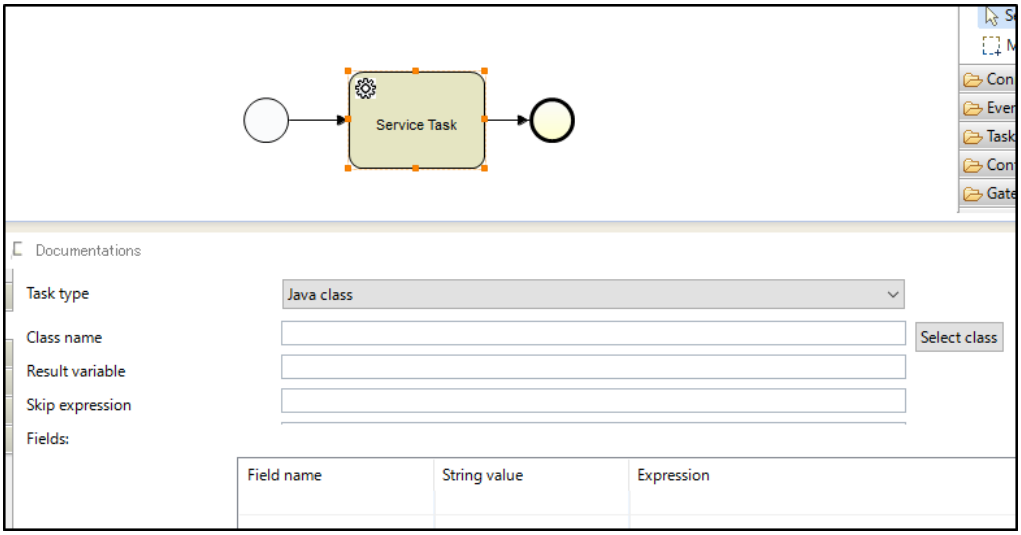
> **コラム**
>
> 結果変数名の設定の詳細は「IM-BPM Designer 操作ガイド」-「スクリプトタスク」を確認してください。

サービスタスク

項目
- javaプログラム
- EL式

javaプログラム

サービスタスクは、javaプログラムを指定して実行することができます。



図：サービスタスク

javaプログラムにはいくつかの制約があります。

- jp.co.intra_mart.activiti.engine.delegate.JavaDelegate インタフェース、もしくは jp.co.intra_mart.activiti.engine.impl.pvm.delegate.ActivityBehavior インタフェースを実装している必要があります。
- デフォルトコンストラクタが存在しなければなりません。
- サービスタスクにフィールドを設定した場合、そのフィールド名の jp.co.intra_mart.activiti.engine.delegate.Expression のクラス変数を宣言している必要があります。

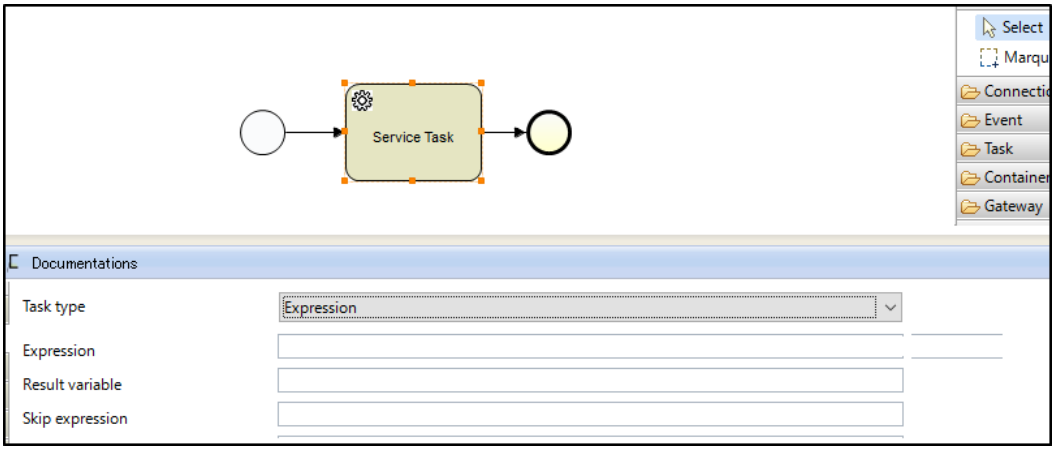
> **コラム**
>
> javaプログラムの設定やフィールドの設定の詳細は「IM-BPM Designer 操作ガイド」-「サービスタスク」を確認してください。

```
package sample;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.Expression;
import jp.co.intra_mart.activiti.engine.delegate.JavaDelegate;

public class SampleJavaDelegate implements JavaDelegate {

    protected Expression param1;
    protected Expression param2;

    @Override
    public void execute(DelegateExecution execution) throws Exception {

        // 変数を取得する。
        Object variable1 = execution.getVariable("variable1");
        Object variable2 = execution.getVariable("variable2");

        // 変数を更新する。変数が存在する場合は更新される。
        execution.setVariable("variable1", ((int) variable1) + (Integer.parseInt((String) param1.getValue(execution))));

        // 変数を追加する。変数が存在しない場合は追加される。
        execution.setVariable("variable3", ((int) variable2) + (Integer.parseInt((String) param2.getValue(execution))));
    }
}
```

EL式

サービスタスクは、EL式の結果を変数に登録することができます。

- タスクタイプを「式」に設定し、EL式を定義します。
- 結果変数名に任意の文字列を設定します。



図：サービスタスク

```
${variable1 + variable2}
```

暗黙オブジェクトとして、executionが存在するため下記のように変数を設定することも可能です。

```
${execution.setVariable('key', 'value')}
```

> **コラム**
>
> タスクタイプの設定や結果変数名の設定の詳細は「IM-BPM Designer 操作ガイド」-「サービスタスク」を確認してください。

リスナ

全てのアクティビティ(シーケンスフローも含む)にリスナを設定することができます。
リスナを設定することにより、任意の処理をアクティビティの開始時、通過時、終了時に実行することができます。
プロセスインスタンスの開始時および終了時も設定することができます。

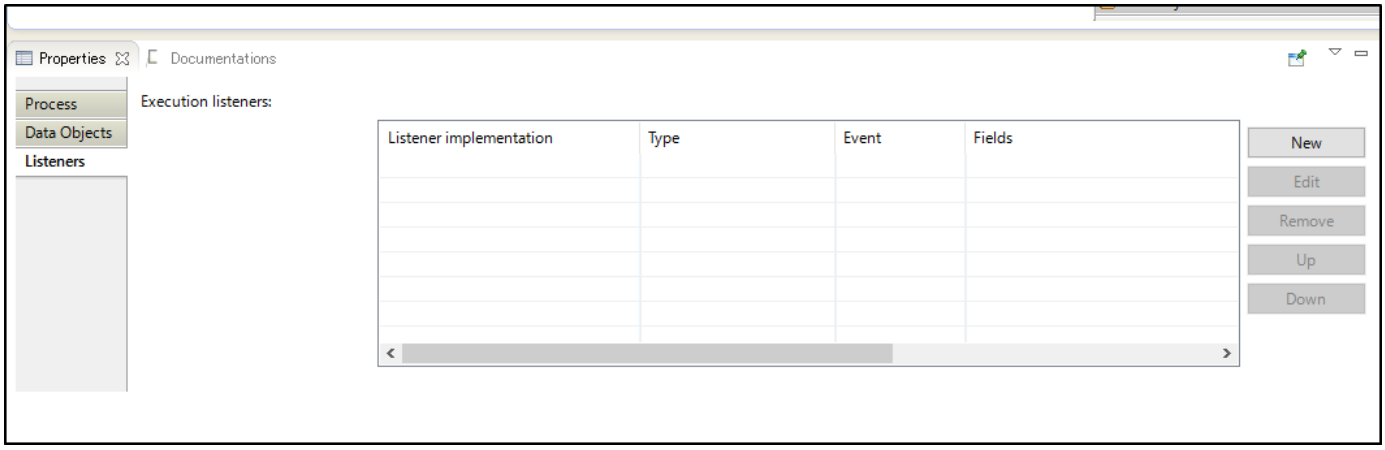


図:リスナ

項目

- javaプログラム
- EL式
- スクリプト
- ユーザタスクでのリスナ

javaプログラム

任意のjavaプログラムを指定して実行することができます。

javaプログラムにはいくつかの制約があります。

- `jp.co.intra_mart.activiti.engine.delegate.ExecutionListener` インタフェースを実装している必要があります。
- デフォルトコンストラクタが存在しなければなりません。
- リスナにフィールドを設定した場合、そのフィールド名の `jp.co.intra_mart.activiti.engine.delegate.Expression` のクラス変数を宣言している必要があります。

> **コラム**
>
> javaプログラムの設定やフィールドの設定の詳細は「IM-BPM Designer 操作ガイド」-「リスナ」を確認してください。

```
package sample;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ExecutionListener;
import jp.co.intra_mart.activiti.engine.delegate.Expression;

public class SampleListener implements ExecutionListener {

    private static final long serialVersionUID = 1L;

    protected Expression param1;
    protected Expression param2;

    @Override
    public void notify(DelegateExecution execution) throws Exception {

        // 変数を取得する。
        Object variable1 = execution.getVariable("variable1");
        Object variable2 = execution.getVariable("variable2");

        // 変数を更新する。変数が存在する場合は更新される。
        execution.setVariable("variable1", ((int) variable1) + (Integer.parseInt((String) param1.getValue(execution))));

        // 変数を追加する。変数が存在しない場合は追加される。
        execution.setVariable("variable3", ((int) variable2) + (Integer.parseInt((String) param2.getValue(execution))));
    }
}
```

EL式

リスナは、EL式を実行することができます。

- タイプを「式」に設定し、EL式を定義します。

```
${execution.setVariable("foo", "FOO")}
```

> **コラム**
>
> タイプの設定の詳細は「IM-BPM Designer 操作ガイド」-「リスナ」を確認してください。

スクリプト

リスナは、スクリプトを実行することができます。

- タイプは「Javaクラス」に設定し、サービスクラスに jp.co.intra_mart.activiti.engine.impl.bpmn.listener.ScriptExecutionListener を指定します。
- フィールドに、フィールド名「language」値「im_javascript」を設定します。
- フィールドに、フィールド名「script」値にスクリプトを設定します。
- フィールドに任意で、フィールド名「resultVariable」値「%任意の文字列%」を設定します。設定した場合、変数にスクリプトの戻り値が%任意の文字列%で作成されます。

```
function run(variables, execution, entity) {entity.setVariable('foo', 'FOO');}
```

ユーザタスクでのリスナ

ユーザタスクにはアクティビティのリスナの他にユーザタスクに関するリスナを設定することができます。
リスナを設定することにより、任意の処理をユーザタスクの作成時、振り分け時、完了時に実行することができます。

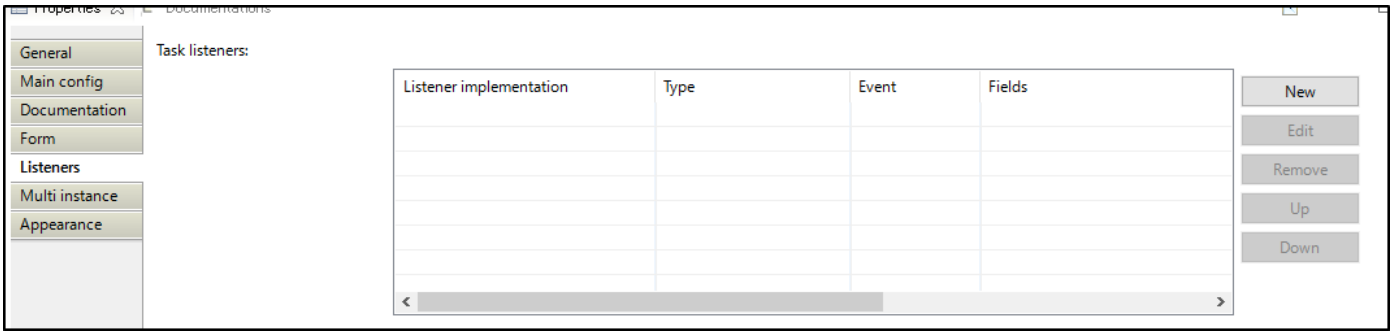


図:リスナ

アクティビティと同様に任意のjavaプログラムとEL式とスクリプトを実行することができます。

javaプログラムを実行する場合は、`jp.co.intra_mart.activiti.engine.delegate.TaskListener` インタフェースを実装してください。

```
package sample;

import jp.co.intra_mart.activiti.engine.delegate.DelegateTask;
import jp.co.intra_mart.activiti.engine.delegate.Expression;
import jp.co.intra_mart.activiti.engine.delegate.TaskListener;

public class SampleTaskListener implements TaskListener {

    private static final long serialVersionUID = 1L;

    protected Expression param1;
    protected Expression param2;

    @Override
    public void notify(DelegateTask task) {

        // 変数を取得する。
        Object variable1 = task.getVariable("variable1");
        Object variable2 = task.getVariable("variable2");

        // 変数をローカルに追加する。
        task.setVariableLocal("variable1", ((int) variable1) + (Integer.parseInt((String) param1.getValue(task))));

        // 変数をローカルに追加する。
        task.setVariableLocal("variable3", ((int) variable2) + (Integer.parseInt((String) param2.getValue(task))));
    }
}
```

スクリプトを実行する場合は、サービスクラスに `jp.co.intra_mart.activiti.engine.impl.bpmn.listener.ScriptTaskListener` を指定してください。

> **注意**
>
> ユーザタスクのリスナのスクリプトの実行は、現在利用できません。

サービスを使用してのプロセスの操作方法

ここでは IM-BPMのサービスを使用してのプロセスの操作方法について説明します。

プロセスインスタンス

プロセスインスタンスは、デプロイされたプロセス定義を開始することにより作成されます。

> **コラム**
>
> デプロイやプロセス定義の詳細は「IM-BPM 仕様書」を確認してください。

項目

プロセスインスタンスを開始する

プロセスインスタンスを開始する方法はいくつかあります。

- プロセス定義キーを指定してプロセスを開始する。
- プロセス定義IDを指定してプロセスを開始する。
- メッセージを指定してプロセスを開始する。

プロセス定義キーを指定してプロセスインスタンスを開始する

プロセス定義キーを指定してプロセスインスタンスを開始します。
プロセス定義キーを指定してプロセスインスタンスを開始した場合、最新バージョンのプロセス定義が開始されます。

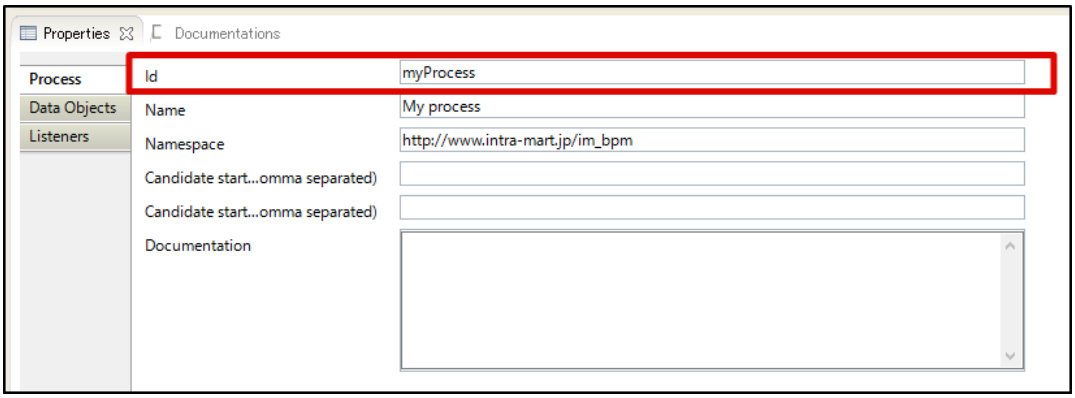


図:プロセス定義キー

> **コラム**
>
> プロセス定義キーの詳細は「IM-BPM 仕様書」-「プロセス」を確認してください。

REST-API

メソッド	POST
URI	%ベースURL%/runtime/process-instances
BODY	{‘processDefinitionKey’ : ‘%プロセス定義キー%’, ‘businessKey’ : ‘%業務キー%’, ‘returnVariables’ : true, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.startProcessInstanceByKey("%プロセス定義キー%");

// 業務キーを設定する場合
runtimeService.startProcessInstanceByKey("%プロセス定義キー%", "%業務キー%");

// 変数を設定する場合
runtimeService.startProcessInstanceByKey("%プロセス定義キー%", "%業務キー%", %変数Map%);
```

プロセス定義IDを指定してプロセスインスタンスを開始する

プロセス定義IDを指定してプロセスインスタンスを開始します。
プロセス定義IDを指定することにより、過去のバージョンのプロセス定義も開始することができます。

コラム

プロセス定義IDの詳細は「IM-BPM 仕様書」-「プロセス」を確認してください。

REST-API

メソッド	POST
URI	%ベースURL%/runtime/process-instances
BODY	{‘processDefinitionId’ : ‘%プロセス定義ID%’, ‘businessKey’ : ‘%業務キー%’, ‘returnVariables’ : true, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.startProcessInstanceById("%プロセス定義ID%");

// 業務キーを設定する場合
runtimeService.startProcessInstanceById("%プロセス定義ID%", "%業務キー%");

// 変数を設定する場合
runtimeService.startProcessInstanceById("%プロセス定義ID%", "%業務キー%", %変数Map%);
```

メッセージを指定してプロセスインスタンスを開始する

メッセージを指定してプロセスインスタンスを開始します。

「 *メッセージ* 」-「 *プロセスインスタンスを開始する* 」を参照してください。

タスク

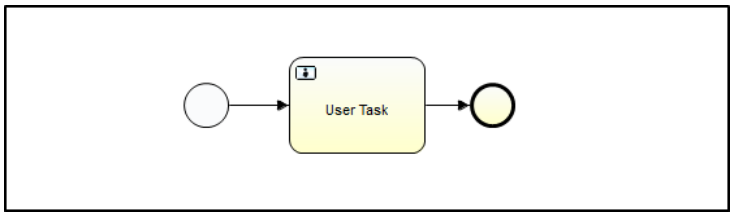


図:タスク(ユーザタスク)

> **コラム**
>
> タスクの詳細は「IM-BPM 仕様書」-「タスク」を確認してください。

タスクを操作する

タスクの操作について以下を解説します。

- タスクを完了させる。
- タスクの担当者を振り分ける。
- タスクの担当者をはずす。

タスクを完了させる

タスクをタスクIDを指定して完了させます。

REST-API

メソッド	POST
URI	%ベースURL%/runtime/tasks/{taskId}
BODY	{'action' : 'complete', 'variables' : [{'name' : '%変数名%', 'type' : '%変数タイプ%', 'variableScope' : '%変数スコープ%', 'value' : '%値%'}, ...]}

API

```
TaskService taskService = ProcessEngineFactory.getInstance().getProcessEngine().getTaskService();

taskService.complete("%タスクID%");

// 変数を設定する場合
taskService.complete("%タスクID%", %変数Map%);
```

タスクの担当者を振り分ける

タスクにタスクIDを指定して担当者を振り分けます。

REST-API

メソッド	POST
URI	%ベースURL%/runtime/tasks/{taskId}
BODY	{‘action’ : ‘claim’, ‘assignee’ : ‘%担当者ID%’}

API

```
TaskService taskService = ProcessEngineFactory.getInstance().getProcessEngine().getTaskService();

taskService.claim("%タスクID%", "%担当者ID%");
```

タスクの担当者をはずす

タスクからタスクIDを指定して担当者をはずします。

REST-API

メソッド	POST
URI	%ベースURL%/runtime/tasks-unclaim/{taskId}
BODY	{}

API

```
TaskService taskService = ProcessEngineFactory.getInstance().getProcessEngine().getTaskService();

taskService.unclaim("%タスクID%");
```

メッセージ

メッセージは、メッセージ開始イベントやメッセージ中間イベントなどに送信することによりプロセスインスタンスを開始および進めることができます。

項目

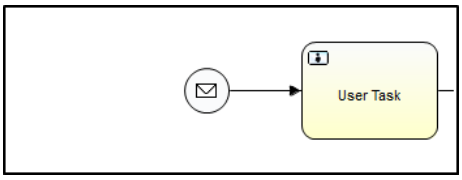
メッセージを送信する

メッセージを送信する用途はいくつかあります。

- プロセス定義のメッセージ開始イベントに送信する。
- メッセージ中間イベントにとまっているプロセスインスタンスに送信する。
- イベントサブプロセスに遷移させるために送信する。
- メッセージ境界イベントを発火させるために送信する。

プロセスインスタンスを開始する

プロセス定義のメッセージ開始イベントに設定している参照メッセージを指定してメッセージを送信します。



図：メッセージ開始イベント

> **コラム**
>
> メッセージ開始イベントの設定の詳細は「IM-BPM Designer 操作ガイド」-「メッセージ開始イベント」を確認してください。

REST-API

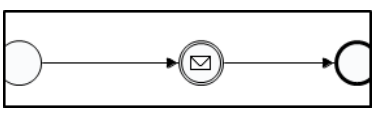
メソッド	POST
URI	%ベースURL%/api/bpm/runtime/process-instances
BODY	{‘message’ : ‘%参照メッセージ%’, ‘businessKey’ : ‘%業務キー%’, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.startProcessInstanceByMessage("%参照メッセージ%");

// 業務キーを設定する場合
runtimeService.startProcessInstanceByMessage("%参照メッセージ%", "%業務キー%");

// 変数を設定する場合
runtimeService.startProcessInstanceByMessage("%参照メッセージ%", %変数Map%);
```

メッセージ中間イベントにとまっているプロセスインスタンスを進める

メッセージ中間イベントに設定している参照メッセージとエグゼキューションIDを指定してメッセージを送信します。



図：メッセージ中間イベント

> **コラム**
>
> メッセージ中間イベントの設定の詳細は「IM-BPM Designer 操作ガイド」-「メッセージ中間イベント」を確認してください。

REST-API

メソッド	PUT
URI	%ベースURL%/runtime/executions/{executionId}
BODY	{‘action’ : ‘messageEventReceived’, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.messageEventReceived("%参照メッセージ%", "%エグゼキューションID%");

// 変数を設定する場合
runtimeService.messageEventReceived("%参照メッセージ%", "%エグゼキューションID%", %変数Map%);
```

イベントサブプロセスに遷移させる

イベントサブプロセスのメッセージ開始イベントに設定している参照メッセージとプロセスインスタンスIDを指定してメッセージを送信します。

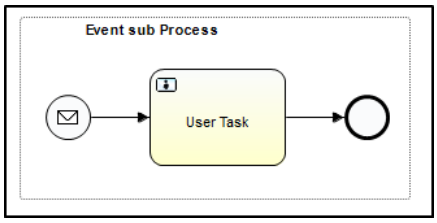


図:イベントサブプロセス - メッセージ開始イベント

> **コラム**
>
> イベントサブプロセスの設定の詳細は「IM-BPM Designer 操作ガイド」-「イベントサブプロセス」を確認してください。

REST-API

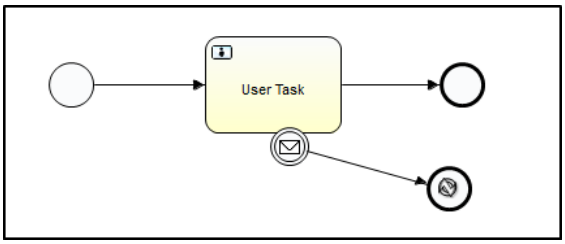
メソッド	PUT
URI	%ベースURL%/runtime/executions/{executionId} * RESTの表記上executionIdになっていますが、プロセスインスタンスIDを設定してください。
BODY	{‘action’ : ‘messageEventReceived’, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.messageEventReceived("%参照メッセージ%", "%プロセスインスタンスID%");

// 変数を設定する場合
runtimeService.messageEventReceived("%参照メッセージ%", "%プロセスインスタンスID%", %変数Map%);
```

メッセージ境界イベントを発火させる

メッセージ境界イベントに設定しているメッセージ参照とエグゼキューションIdを指定してメッセージを送信します。



図：メッセージ境界イベント

> **コラム**
>
> メッセージ境界イベントの設定の詳細は「IM-BPM Designer 操作ガイド」-「メッセージ境界イベント」を確認してください。

メッセージ中間イベントにとまっているプロセスインスタンスを進める方法と同様です。

シグナル

シグナルは、シグナル中間イベントなどのシグナルの受信を待機しているプロセスインスタンスを進めることができます。

シグナルを送信する

シグナルを送信する用途はいくつかあります。

- シグナル中間イベントにとまっているプロセスインスタンスにブロードキャストする。
- シグナル境界イベントを発火させるためにブロードキャストする。
- レシーブタスクに送信する。

シグナル中間イベントにとまっているプロセスインスタンスを進める

シグナル中間イベントに設定している参照シグナルを指定してシグナルをブロードキャストします。

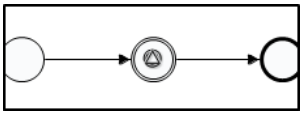


図:シグナル中間イベント

> **コラム**
>
> シグナル中間イベントの設定の詳細は「IM-BPM Designer 操作ガイド」-「シグナル中間イベント」を確認してください。

REST-API

メソッド	POST
URI	%ベースURL%/runtime/signals
BODY	{‘signalName’ : ‘%参照シグナル%’, ‘async’ : false, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.signalEventReceived("%参照シグナル%");

// 変数を設定する場合
runtimeService.signalEventReceived("%参照シグナル%", %変数Map%);

// 非同期でシグナルを送信する場合
runtimeService.signalEventReceivedAsync("%参照シグナル%");
```

プロセスインスタンスを指定してシグナルを送信する場合

REST-API

メソッド	PUT
URI	%ベースURL%/runtime/executions/{executionId}
BODY	{‘action’ : ‘signalEventReceived’, ‘signalName’ : ‘%参照シグナル%’, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

参照シグナルを指定しない場合

メソッド	PUT
URI	%ベースURL%/runtime/executions/{executionId}
BODY	{‘action’ : ‘signal’, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

```
RuntimeService runtimeService = ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

runtimeService.signalEventReceived("%参照シグナル%", "%エグゼキューションID%");

// 変数を設定する場合
runtimeService.signalEventReceived("%参照シグナル%", "%エグゼキューションID%", %変数Map%);

// 参照シグナルを指定しない場合
runtimeService.signal("%エグゼキューションID%");
```

シグナル境界イベントを発火させるためにブロードキャストする

シグナル境界イベントに設定しているシグナル参を指定してシグナルをブロードキャストします。

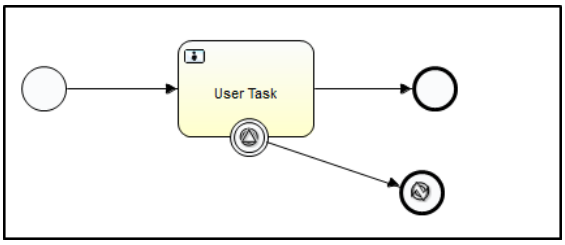


図:シグナル境界イベント

> **コラム**
>
> シグナル境界イベントの設定の詳細は「IM-BPM Designer 操作ガイド」-「シグナル境界イベント」を確認してください。

シグナル中間イベントにとまっているプロセスインスタンスを進める方法と同様です。

レシーブタスクに送信する

レシーブタスクにとまっているプロセスのエグゼキューションIDを指定してメッセージを送信します。

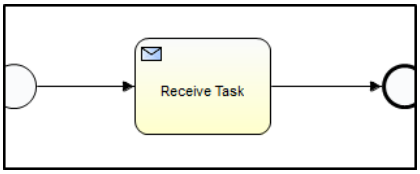


図:レシーブタスク

> **コラム**
>
> レシーブタスクの設定の詳細は「IM-BPM Designer 操作ガイド」-「レシーブタスク」を確認してください。

シグナル中間イベントにとまっているプロセスインスタンスを進める方法の「参照シグナルを指定しない場合」と同様です。

他アプリケーションとの連携方法

ここでは IM-BPMの他アプリケーションとの連携方法について説明します。

IM-Workflow

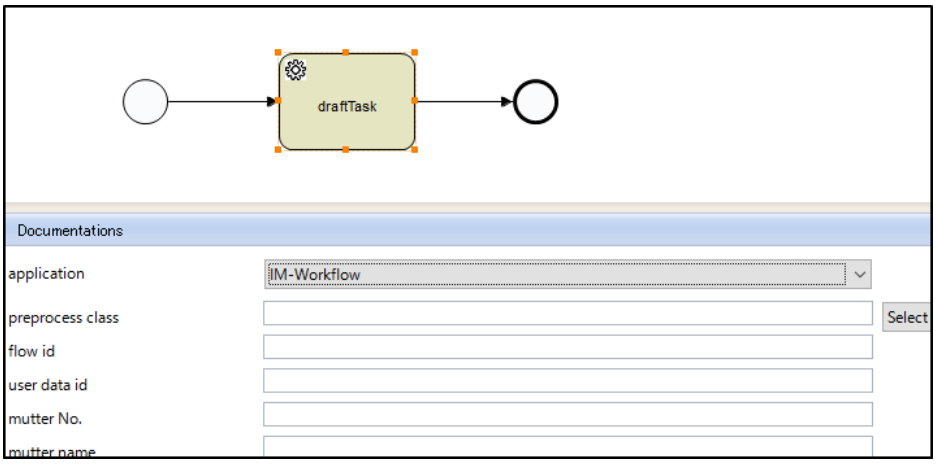
IM-BPMから、IM-Workflowとの連携方法を説明します。

項目

- 起票・申請タスクに前処理ユーザプログラムを設定する

起票・申請タスクに前処理ユーザプログラムを設定する

IM-Workflowを起票または申請する前の処理を差し込みます。



図：起票タスク

> **コラム**
>
> 前処理ユーザプログラムの設定の詳細は「IM-BPM Designer 操作ガイド」-「起票タスク」を確認してください。

前処理ユーザプログラムは、決められたインタフェースを実装する必要があります。

- 起票の場合 - jp.co.intra_mart.activiti.engine.delegate.ImWorkflowDraftPreprocess
- 申請の場合 - jp.co.intra_mart.activiti.engine.delegate.ImWorkflowApplyPreprocess

```
package sample;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ImWorkflowDraftPreprocess;
import jp.co.intra_mart.foundation.workflow.application.model.param.DraftParam;

public class SampleWorkflowDraftPreprocess implements ImWorkflowDraftPreprocess {

   @Override
   public void execute(DelegateExecution execution, DraftParam param)
         throws Exception {
      String matterName = (String) execution.getVariable("matterName");
      String matterNumber = (String) execution.getVariable("matterNumber");

      if (matterName != null) {
         param.setMatterName(matterName);
      }

      if (matterNumber != null) {
         param.setMatterNumber(matterNumber);
      }
   }
}
```

```
package sample;

import java.util.Map;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ImWorkflowApplyParamModel;
import jp.co.intra_mart.activiti.engine.delegate.ImWorkflowApplyPreprocess;
import jp.co.intra_mart.foundation.workflow.application.model.param.ApplyParam;

public class SampleWorkflowApplyPreprocess implements ImWorkflowApplyPreprocess {

  @Override
  public void execute(DelegateExecution execution, ImWorkflowApplyParamModel model)
      throws Exception {
    ApplyParam applyParam = model.getApplyParam();

    String matterName = (String) execution.getVariable("matterName");
    String matterNumber = (String) execution.getVariable("matterNumber");

    if (matterName != null) {
      applyParam.setMatterName(matterName);
    }

    if (matterNumber != null) {
      applyParam.setMatterNumber(matterNumber);
    }

    Map<String, Object> userParam =  model.getUserParam();

    String param1 = (String) execution.getVariable("param1");
    String param2 = (String) execution.getVariable("param2");

    userParam.put("param1", param1);
    userParam.put("param2", param2);
  }
}
```

IM-FormaDesigner

IM-BPMから、IM-FormaDesignerとの連携方法を説明します。
IM-BPMから、IM-FormaDesignerと連携する場合、jugglingプロジェクトにてアプリケーションを追加する必要があります。

項目

- 前処理、後処理をカスタマイズする
- 起票・申請タスクに前処理ユーザプログラムを設定する

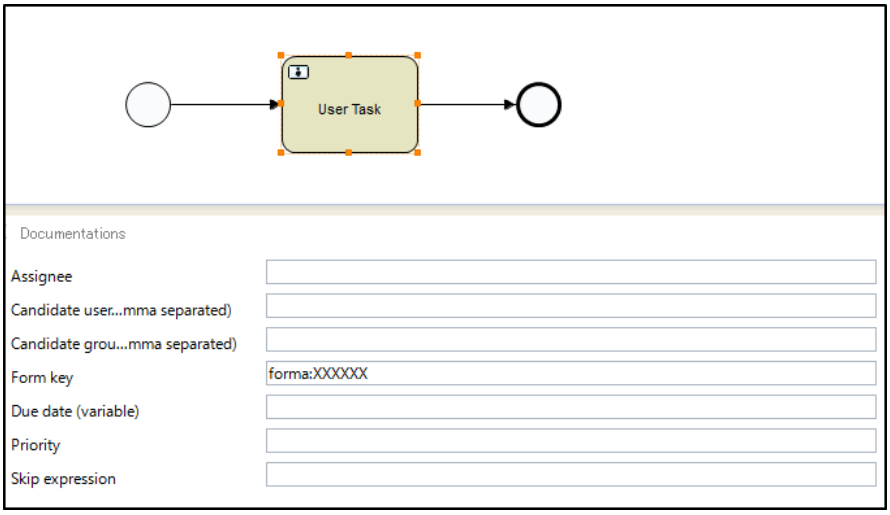
前処理、後処理をカスタマイズする

IM-FormaDesignerの前処理、後処理をカスタマイズします。

コラム

IM-FormaDesignerの前処理、後処理の詳細は「IM-FormaDesigner 作成者操作ガイド」を確認してください。

IM-BPMのユーザタスクの From key にIM-FormaDesignerのIDを指定することでIM-FormaDesignerと連携することができます。



図：From key

> **コラム**
>
> ユーザタスクの From key についての詳細は「IM-BPM Designer 操作ガイド」-「ユーザタスク」を確認してください。

プロセス定義をデプロイし時点で、ユーザタスクに自動でIM-FormaDesignerに対して、前処理、後処理が設定されます。

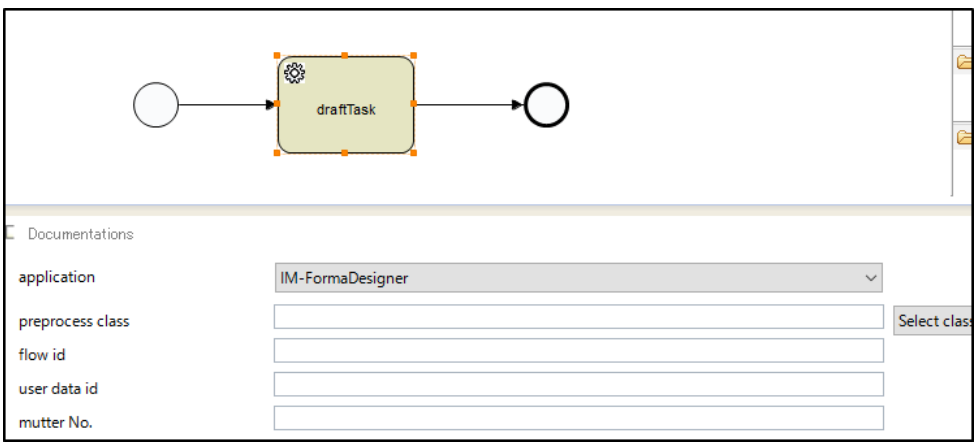
- 前処理 - jp.co.intra_mart.activiti.extension.forma.BPMPreProcessExecutor
- 後処理 - jp.co.intra_mart.activiti.extension.forma.BPMPostProcessExecutor

前処理の主な処理は、プロセスインスタンスとタスクの変数を取得して画面の初期値に設定しています。
後処理の主な処理は、プロセスインスタンスを開始したり、タスクを完了し画面で入力された値を変数に登録しています。

前処理、後処理を変更したい場合は、jp.co.intra_mart.activiti.extension.forma.ActivitiPreProcessExecutor を継承したクラスを作成し
IM-FormaDesignerの機能に従い、ユーザプログラム一覧から既存のクラスを削除し作成したクラスを追加してください。

起票・申請タスクに前処理ユーザプログラムを設定する

IM-FormaDesignerを起票または申請する前の処理を差し込みます。



図：起票タスク

コラム

前処理ユーザプログラムの設定の詳細は「IM-BPM Designer 操作ガイド」-「起票タスク」を確認してください。

前処理ユーザプログラムは、決められたインタフェースを実装する必要があります。

- 起票の場合 - jp.co.intra_mart.activiti.engine.delegate.ImFormaDraftPreprocess
- 申請の場合 - jp.co.intra_mart.activiti.engine.delegate.ImFormaApplyPreprocess

```
package sample;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ImFormaDraftPreprocess;
import jp.co.intra_mart.foundation.workflow.application.model.param.DraftParam;

public class SampleFormaDraftPreprocess implements ImFormaDraftPreprocess {

  @Override
  public void execute(DelegateExecution execution, DraftParam param)
      throws Exception {
    String matterName = (String) execution.getVariable("matterName");
    String matterNumber = (String) execution.getVariable("matterNumber");

    if (matterName != null) {
      param.setMatterName(matterName);
    }

    if (matterNumber != null) {
      param.setMatterNumber(matterNumber);
    }
  }
}
```

```
package sample;

import java.util.Map;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ImFormaApplyParamModel;
import jp.co.intra_mart.activiti.engine.delegate.ImFormaApplyPreprocess;
import jp.co.intra_mart.foundation.workflow.application.model.param.ApplyParam;
import jp.co.intra_mart.foundation.forma.imw.api.type.impl.StandardFormaUserParamKey;

public class SampleFormaApplyPreprocess implements ImFormaApplyPreprocess {

    @Override
    public void execute(DelegateExecution execution, ImFormaApplyParamModel model)
            throws Exception {
        ApplyParam applyParam = model.getApplyParam();

        String matterName = (String) execution.getVariable("matterName");
        String matterNumber = (String) execution.getVariable("matterNumber");

        if (matterName != null) {
            applyParam.setMatterName(matterName);
        }

        if (matterNumber != null) {
            applyParam.setMatterNumber(matterNumber);
        }

        Map<String, Object> itemsMap = model.getFormaUserParam.get(StandardFormaUserParamKey.ITEMS);

        String param1 = (String) execution.getVariable("param1");
        String param2 = (String) execution.getVariable("param2");

        itemsMap.put("param1", param1);
        itemsMap.put("param2", param2);
    }
}
```

IM-BIS

IM-BPMから、IM-BISとの連携方法を説明します。
IM-BPMから、IM-BISと連携する場合、jugglingプロジェクトにてアプリケーションを追加する必要があります。

項目
- 起票・申請タスクに前処理ユーザプログラムを設定する

起票・申請タスクに前処理ユーザプログラムを設定する

IM-BISを起票または申請する前の処理を差し込みます。

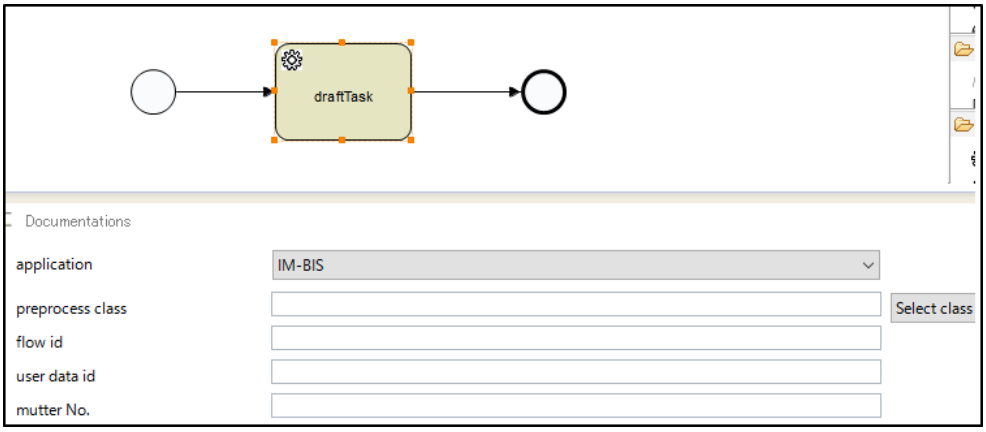


図:起票タスク

> **コラム**
>
> 前処理ユーザプログラムの設定の詳細は「IM-BPM Designer 操作ガイド」-「起票タスク」を確認してください。

前処理ユーザプログラムは、決められたインタフェースを実装する必要があります。

- 起票の場合 - jp.co.intra_mart.activiti.engine.delegate.ImBisDraftPreprocess
- 申請の場合 - jp.co.intra_mart.activiti.engine.delegate.ImBisApplyPreprocess

```
package sample;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ImBisDraftPreprocess;
import jp.co.intra_mart.foundation.workflow.application.model.param.DraftParam;

public class SampleBisDraftPreprocess implements ImBisDraftPreprocess {

   @Override
   public void execute(DelegateExecution execution, DraftParam param)
         throws Exception {
      String matterName = (String) execution.getVariable("matterName");
      String matterNumber = (String) execution.getVariable("matterNumber");

      if (matterName != null) {
         param.setMatterName(matterName);
      }

      if (matterNumber != null) {
         param.setMatterNumber(matterNumber);
      }
   }
}
```

```
package sample;

import java.util.Map;

import jp.co.intra_mart.activiti.engine.delegate.DelegateExecution;
import jp.co.intra_mart.activiti.engine.delegate.ImBisApplyParamModel;
import jp.co.intra_mart.activiti.engine.delegate.ImBisApplyPreprocess;
import jp.co.intra_mart.foundation.workflow.application.model.param.ApplyParam;
import jp.co.intra_mart.foundation.forma.imw.api.type.impl.StandardFormaUserParamKey;

public class SampleBisApplyPreprocess implements ImBisApplyPreprocess {

  @Override
  public void execute(DelegateExecution execution, ImBisApplyParamModel model)
      throws Exception {
    ApplyParam applyParam = model.getApplyParam();

    String matterName = (String) execution.getVariable("matterName");
    String matterNumber = (String) execution.getVariable("matterNumber");

    if (matterName != null) {
      applyParam.setMatterName(matterName);
    }

    if (matterNumber != null) {
      applyParam.setMatterNumber(matterNumber);
    }

    Map<String, Object> itemsMap = model.getFormaUserParam.get(StandardFormaUserParamKey.ITEMS);

    String param1 = (String) execution.getVariable("param1");
    String param2 = (String) execution.getVariable("param2");

    itemsMap.put("param1", param1);
    itemsMap.put("param2", param2);
  }
}
```