目次

- 1. 改訂情報
- 2. はじめに
  - 2.1. 本書の目的
  - 2.2. 対象読者
  - 2.3. サンプルコードについて
  - 2.4. 本書の構成
- 3. プロセス定義
  - 3.1. スクリプトタスク
    - 3.1.1. スクリプトを作成する
    - 3.1.2. 結果を返却する
  - 3.2. サービスタスク
    - 3.2.1. javaプログラム
    - 3.2.2. EL式
    - 3.2.3. エラーハンドリング
  - 3.3. リスナ
    - 3.3.1. javaプログラム
    - 3.3.2. EL式
    - 3.3.3. スクリプト
    - 3.3.4. ユーザタスクでのリスナ
- 4. サービスを使用してのプロセスの操作方法
  - 4.1. プロセスインスタンス
    - 4.1.1. プロセスインスタンスを開始する
      - 4.1.1.1. 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
        - 4.1.1.1.1. REST-API
        - 4.1.1.1.2. JavaEE開発モデル
        - 4.1.1.1.3. スクリプト開発モデル
      - 4.1.1.2. 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
        - 4.1.1.2.1. REST-API
        - 4.1.1.2.2. JavaEE開発モデル
        - 4.1.1.2.3. スクリプト開発モデル
      - 4.1.1.3. 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
      - 4.1.1.4. シグナルを指定してプロセスインスタンスを開始する
  - 4.2. タスク
    - 4.2.1. タスクを操作する
      - 4.2.1.1. タスクを完了させる
        - 4.2.1.1.1. REST-API
        - 4.2.1.1.2. JavaEE開発モデル
        - 4.2.1.1.3. スクリプト開発モデル
      - 4.2.1.2. タスクの担当者を振り分ける
        - 4.2.1.2.1. REST-API

- 4.2.1.2.2. JavaEE開発モデル
- 4.2.1.2.3. スクリプト開発モデル
- 4.2.1.3. タスクの担当者を外す
- 4.2.1.3.1. REST-API
- 4.2.1.3.2. JavaEE開発モデル
- 4.2.1.3.3. スクリプト開発モデル
- 4.3. メッセージ
- 4.3.1. メッセージを送信する
- 4.3.1.1. プロセスインスタンスを開始する
- 4.3.1.1.1. REST-API
- 4.3.1.1.2. JavaEE開発モデル
- 4.3.1.1.3. スクリプト開発モデル
- 4.3.1.2. メッセージキャッチ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
- 4.3.1.2.1. REST-API
- 4.3.1.2.2. JavaEE開発モデル
- 4.3.1.2.3. スクリプト開発モデル
- 4.3.1.3. イベントサブプロセスに遷移させる
- 4.3.1.3.1. REST-API
- 4.3.1.3.2. API
- 4.3.1.3.3. スクリプト開発モデル
- 4.3.1.4. メッセージ境界イベントを発火させる
- 4.4. シグナル
- 4.4.1. シグナルを送信する
- 4.4.1.1. シグナルキャッチ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
- 4.4.1.1.1. 参照シグナルを指定してシグナルをブロードキャストする
- 4.4.1.1.2. 特定のシグナルキャッチイベントに対してシグナルを送信する
- 4.4.1.2. 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
- 4.4.1.3. 受信タスクに送信する
- 4.4.1.4. プロセスインスタンスを開始する
- 5. 他アプリケーションとの連携方法
- 5.1. IM-Workflow
- 5.1.1. 起票・申請タスク に前処理ユーザプログラムを設定する
- 5.2. IM-FormaDesigner
- 5.2.1. 前処理、後処理をカスタマイズする
- 5.2.2. 起票・申請タスクに前処理ユーザプログラムを設定する
- 5.3. IM-BIS
- 5.3.1. 起票・申請タスクに前処理ユーザプログラムを設定する

改訂情報

変更年月日	変更内容
2016-10-01	初版
2016-12-01	第2版 下記のページにスクリプト開発モデルのコード例を加筆しました。 ・「*プロセスインスタンス*」 ・「*タスク*」 ・「*メッセージ*」 ・「*シグナル*」
2017-04-01	第3版 下記を追加・変更しました。 ・「*リスナ*」からユーザタスクのリスナのスクリプトの実行ができない注意事項を削除しました。 ・「*シグナル*」にプロセスインスタンスの開始方法を加筆しました。
2017-12-01	第4版 下記を追加・変更しました。 ・「*サービスタスク*」にエラーハンドリングの項目を加筆しました。
2018-12-01	第5版 下記を追加・変更しました。 ・「IM-BPM プロセスデザイナ 操作ガイド」に説明を対応しました。

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です。

- IM-BPMを理解している
- プロセスデザイナ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

本書は以下のように構成されています。

- *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*

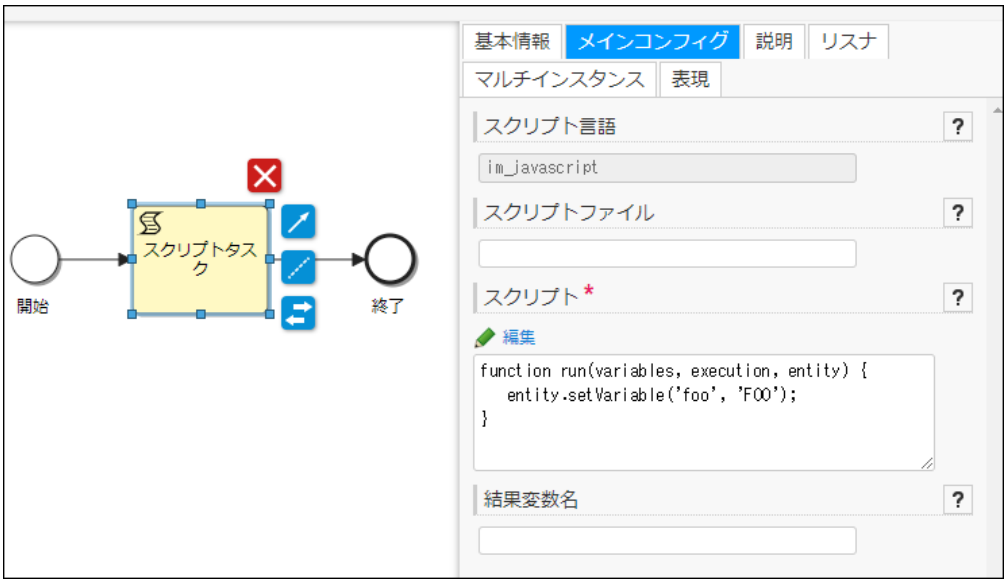
intra-mart Accel Platformのアプリケーションとの連携について説明します。

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

設定されたスクリプトを自動的に実行するタスクです。



図：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、 プロセスデザイナ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。

```
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、以下です。
  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。

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};
}
```

複数の値の返却できます。

```
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};
}
```

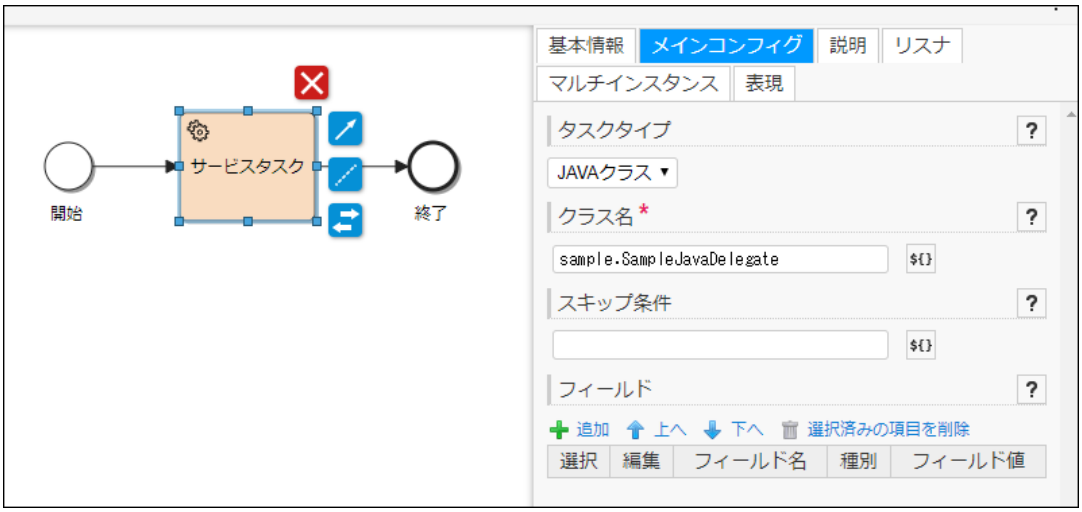
コラム

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 プロセスデザイナ 操作ガイド」 - 「スクリプトタスク」を参照してください。

サービスタスク

サービスタスクは、javaプログラムを指定して実行できます。



図：サービスタスク

項目

- javaプログラム
- EL式
- エラーハンドリング

javaプログラム

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、または、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。

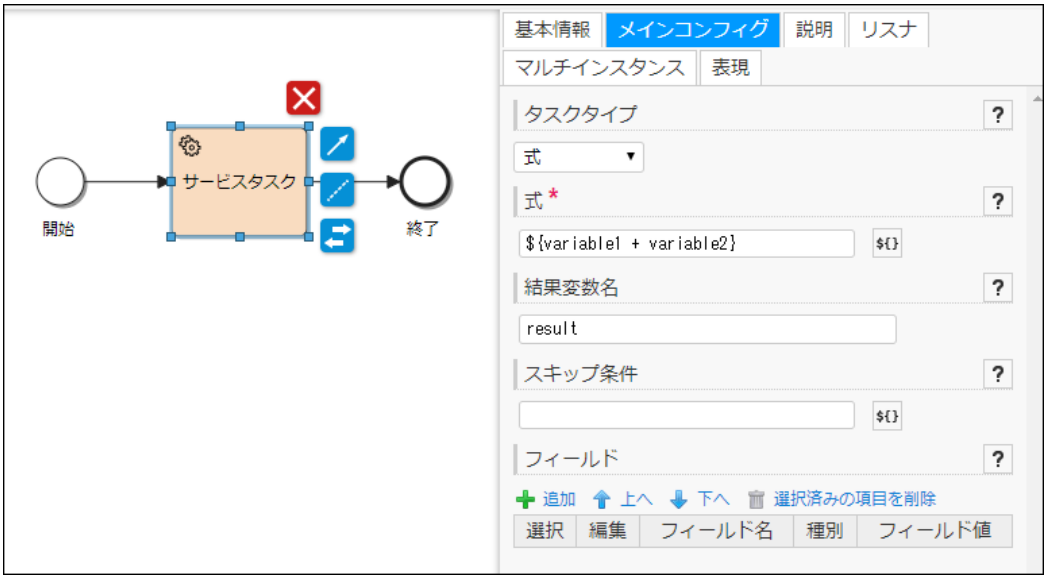
> **コラム**
>
> javaプログラムの設定やフィールドの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「サービスタスク」を参照してください。

```
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。

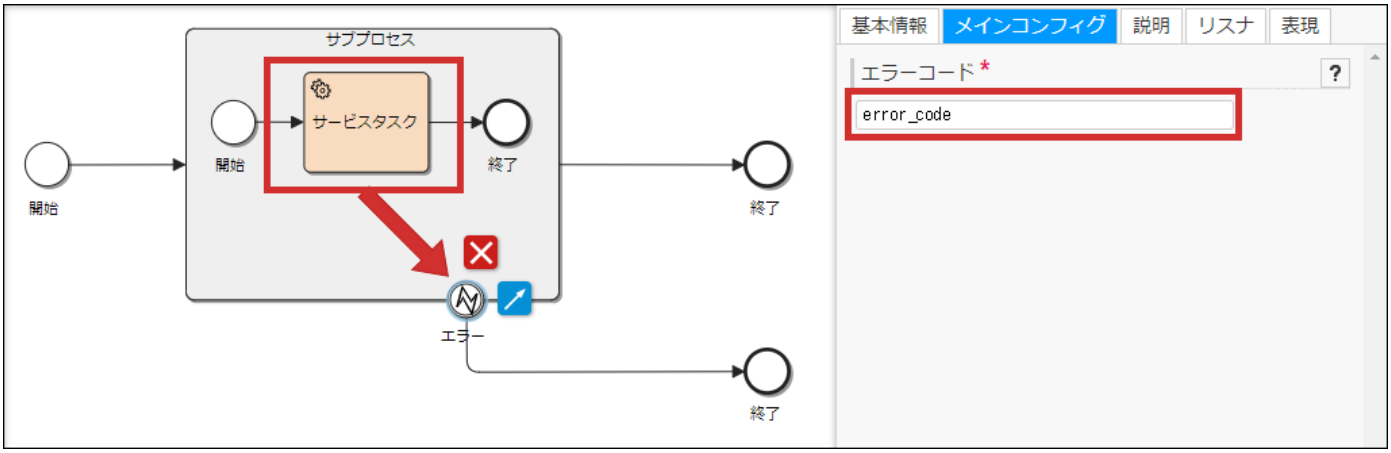
```
${execution.setVariable('key', 'value')}
```

> **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 プロセスデザイナ 操作ガイド」 - 「サービスタスク」を参照してください。

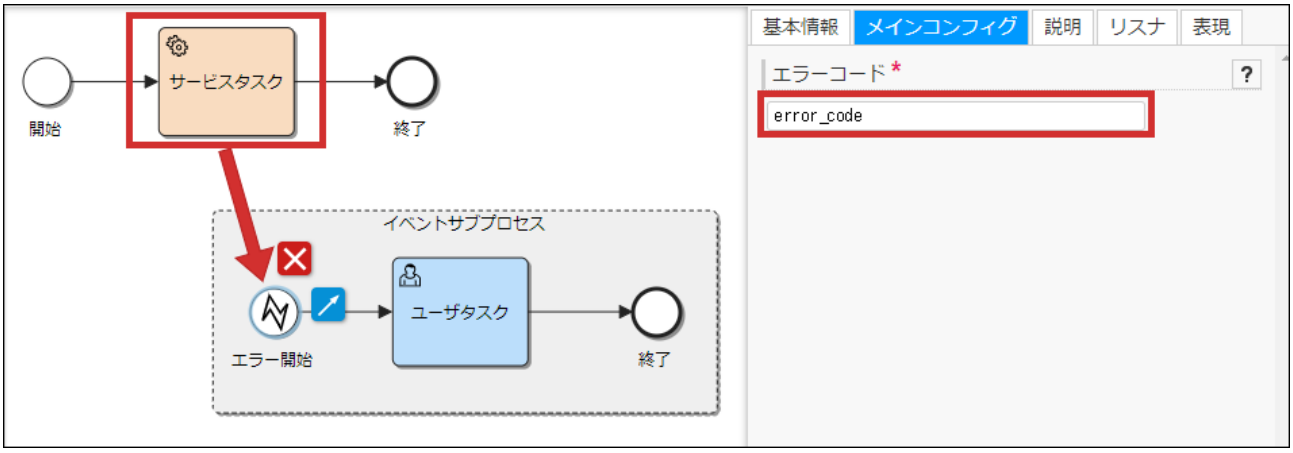
エラーハンドリング

javaプログラムから jp.co.intra_mart.activiti.engine.delegate.BpmnError をスローすることで、以下のイベントを動作させることが可能です。

- エラー境界イベントでキャッチする。
- イベントサブプロセスのエラー開始イベントを開始する。



図：エラー境界イベント



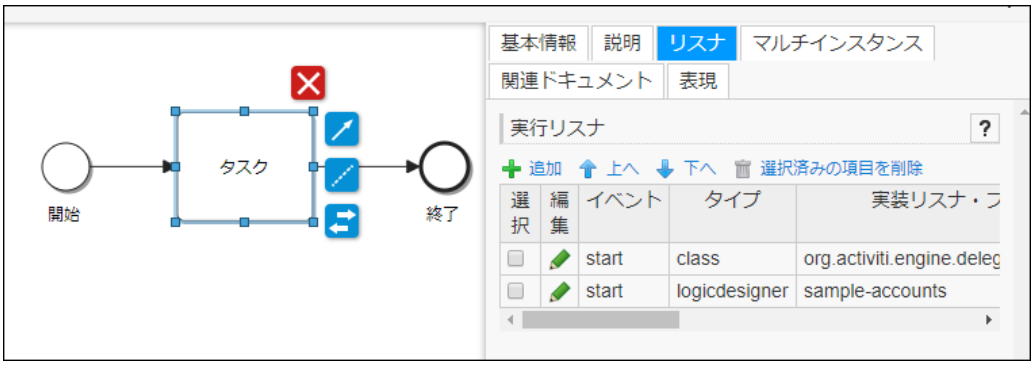
図：エラー開始イベント

> **コラム**
>
> エラー境界イベント、エラー開始イベントの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「エラー境界イベント」「エラー開始イベント」を参照してください。

```
package sample;

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

public class SampleJavaDelegate implements JavaDelegate {

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

      try {

         ...

      } catch (Exception e) {
         // エラーイベントに設定しているエラーコードを指定し、BpmnErrorをスローする。
         throw new BpmnError("error_code");
      }
   }
}
```

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 プロセスデザイナ 操作ガイド」 - 「実行リスナ」を参照してください。

```
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式を実行できます。

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

> **コラム**
>
> タイプの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「実行リスナ」を参照してください。

スクリプト

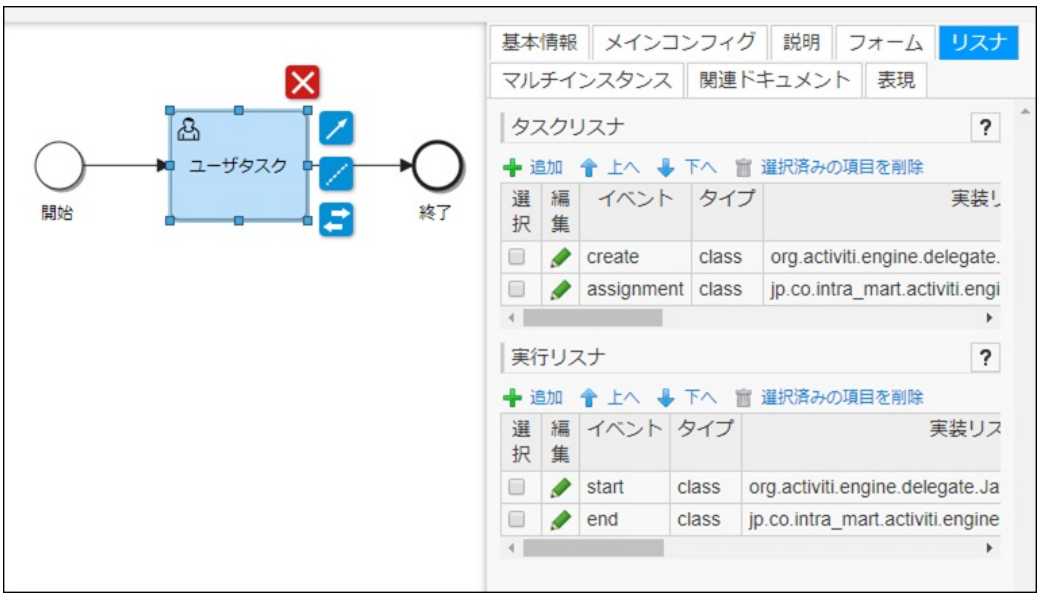
リスナは、スクリプトを実行できます。

- 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。

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」を参照してください。

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ID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する
  - シグナルを指定して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

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」を参照してください。

REST-API

メソッド	POST
URI	%ベースURL%/api/bpm/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%’}, ...]}

JavaEE開発モデル

```
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%);
```

スクリプト開発モデル

```
var runtimeService = new bpm.RuntimeService();

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

// 変数を設定する場合
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.startProcessInstanceByKey("%プロセス定義キー%", variables);

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

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」を参照してください。

REST-API

メソッド	POST
URI	%ベースURL%/api/bpm/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%’}, ...]}

JavaEE開発モデル

```
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%);
```

```
var runtimeService = new bpm.RuntimeService();

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

// 変数を設定する場合
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.startProcessInstanceById("%プロセス定義ID%", variables);

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

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

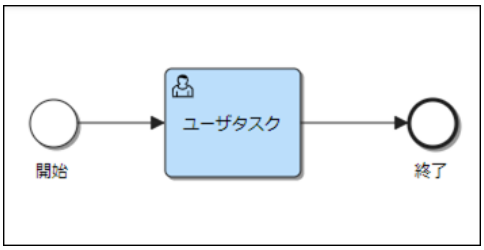
メッセージを指定してプロセスインスタンスを開始します。
「 *メッセージ* 」 - 「 *プロセスインスタンスを開始する* 」を参照してください。

シグナルを指定してプロセスインスタンスを開始する

シグナルを指定してプロセスインスタンスを開始します。
「 *シグナル* 」 - 「 *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

プロセス定義の中での最小単位の業務・アクションを表します。



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」を参照してください。

項目

- タスクを操作する
  - タスクを完了させる
  - タスクの担当者を振り分ける
  - タスクの担当者を外す

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%/api/bpm/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%’}, ...]}

JavaEE開発モデル

```
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%);
```

スクリプト開発モデル

```
var taskService = new bpm.TaskService();

taskService.complete("%タスクID%");

// プロセスの変数を設定する場合
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
taskService.complete("%タスクID%", variables);

// タスクローカルの変数を設定する場合
taskService.complete("%タスクID%", variables, true);
```

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%/api/bpm/runtime/tasks/{taskId}
BODY	{‘action’ : ‘claim’, ‘assignee’ : ‘%担当者ID%’}

JavaEE開発モデル

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

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

スクリプト開発モデル

```
var taskService = new bpm.TaskService();

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

タスクの担当者を外す

タスクからタスクIDを指定して担当者を外します。

REST-API

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

JavaEE開発モデル

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

taskService.unclaim("%タスクID%");
```

スクリプト開発モデル

```
var taskService = new bpm.TaskService();

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 「メッセージキャッチイベント」などに送信することで、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
  - プロセスインスタンスを開始する
  - メッセージキャッチ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
  - イベントサブプロセスに遷移させる
  - メッセージ境界イベントを発火させる

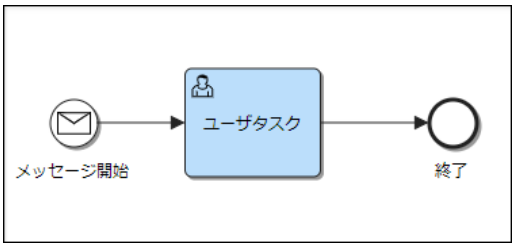
メッセージを送信する

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。
- メッセージキャッチ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された「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 プロセスデザイナ 操作ガイド」 - 「メッセージ開始イベント」を参照してください。

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%’}, ...]}

JavaEE開発モデル

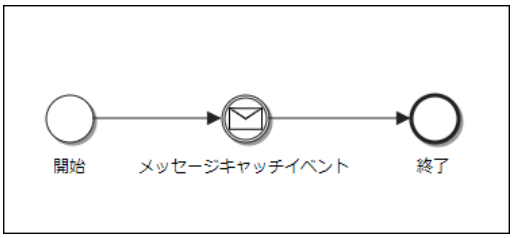
```
RuntimeService runtimeService =
ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

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%);
```

スクリプト開発モデル

```
var runtimeService = new bpm.RuntimeService();

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

// 変数を設定する場合
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.startProcessInstanceByMessage("%参照メッセージ%", variables);

// 業務キーと変数を設定する場合
runtimeService.startProcessInstanceByMessage("%参照メッセージ%", "%業務キー%", variables);
```

メッセージキャッチ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

メッセージキャッチイベントに設定された「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：メッセージキャッチイベント

> **コラム**
>
> メッセージキャッチイベントの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「メッセージキャッチイベント」を参照してください。

REST-API

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

JavaEE開発モデル

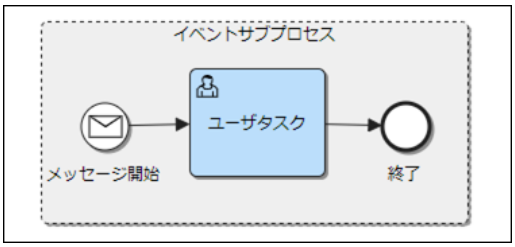
```
RuntimeService runtimeService =
ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

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%);
```

スクリプト開発モデル

```
var runtimeService = new bpm.RuntimeService();

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
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.messageEventReceived("%参照メッセージ%", "%エグゼキューションID%", variables);
```

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 プロセスデザイナ 操作ガイド」 - 「イベントサブプロセス」を参照してください。

REST-API

メソッド	PUT
URI	%ベースURL%/api/bpm/runtime/executions/{executionId} * RESTの表記上はexecutionIdですが、プロセスインスタンスIDを設定してください。
BODY	{‘action’ : ‘messageEventReceived’, ‘messageName’ : ‘%参照メッセージ%’, ‘variables’ : [{‘name’ : ‘%変数名%’, ‘type’ : ‘%変数タイプ%’, ‘variableScope’ : ‘%変数スコープ%’, ‘value’ : ‘%値%’}, ...]}

API

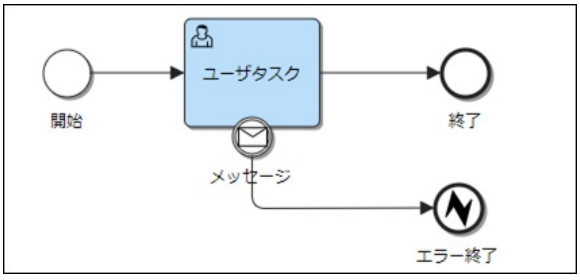
```
RuntimeService runtimeService =
ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

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%);
```

スクリプト開発モデル

```
var runtimeService = new bpm.RuntimeService();

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
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.messageEventReceived("%参照メッセージ%", "%プロセスインスタンスID%", variables);
```

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された「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。
メッセージキャッチ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。



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 プロセスデザイナ 操作ガイド」 - 「メッセージ境界イベント」を参照してください。

シグナル

シグナルは、シグナルキャッチ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。

項目

- シグナルを送信する
  - シグナルキャッチ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
    - 参照シグナルを指定してシグナルをブロードキャストする
    - 特定のシグナルキャッチイベントに対してシグナルを送信する
  - 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
  - 受信タスクに送信する
  - プロセスインスタンスを開始する

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。

- シグナルキャッチ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。
- 受信タスクに送信する。
- プロセス定義のシグナル開始イベントにブロードキャストする。

シグナルキャッチ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

シグナルをブロードキャストする方法と、特定のイベントに対してシグナルを送信する方法があります。

図：シグナルキャッチイベント

> **コラム**
>
> シグナルキャッチイベントの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「シグナルキャッチイベント」を参照してください。

参照シグナルを指定してシグナルをブロードキャストする

REST-API

メソッド	POST
URI	%ベースURL%/api/bpm/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%’}, ...]}

JavaEE開発モデル

```
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%);

// シグナル受信アクティビティを非同期実行する場合
runtimeService.signalEventReceivedAsync("%参照シグナル%");
```

スクリプト開発モデル

```
var runtimeService = new bpm.RuntimeService();

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

// 変数を設定する場合
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.signalEventReceived("%参照シグナル%", variables);

// シグナル受信アクティビティを非同期実行する場合
runtimeService.signalEventReceivedAsync("%参照シグナル%");
```

特定のシグナルキャッチイベントに対してシグナルを送信する

REST-API

- 参照シグナルを指定する場合

メソッド	PUT
URI	%ベースURL%/api/bpm/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%/api/bpm/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%’}, ...]}

JavaEE開発モデル

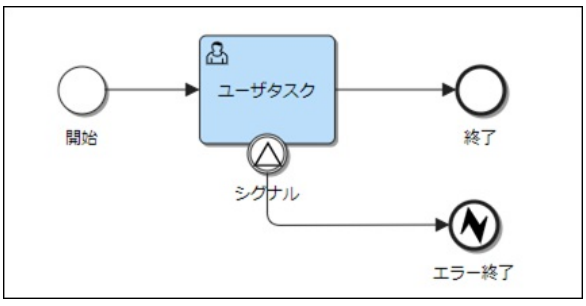
```
RuntimeService runtimeService =
ProcessEngineFactory.getInstance().getProcessEngine().getRuntimeService();

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%");
```

スクリプト開発モデル

```
var runtimeService = new bpm.RuntimeService();

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
var variables = {
  "var1": "string",
  "var2": 123,
  "var3": new Date(),
  "var4": true
};
runtimeService.signalEventReceived("%参照シグナル%", "%エグゼキューションID%", variables);

// 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。



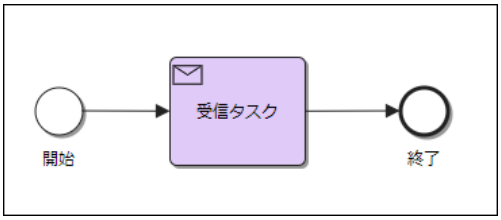
図：シグナル境界イベント

> **コラム**
>
> シグナル境界イベントの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「シグナル境界イベント」を参照してください。

シグナルキャッチイベント 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。

受信タスクに送信する

受信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。
「*シグナルキャッチ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*」 の 「参照シグナルを指定しない場合」と同様の手段です。

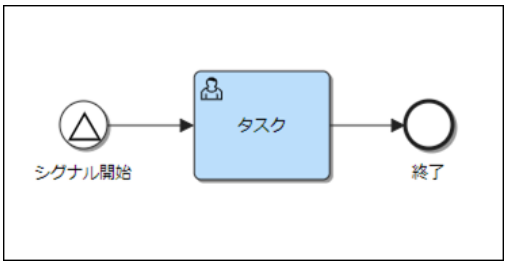


図：受信タスク

> **コラム**
>
> 受信タスクの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「受信タスク」を参照してください。

プロセスインスタンスを開始する

プロセス定義のシグナル開始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。
「*参照シグナルを指定してシグナルをブロードキャストする*」 と同様の手段です。



図：シグナル開始イベント

> **コラム**
>
> シグナル開始イベントの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「シグナル開始イベント」を参照してください。

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。

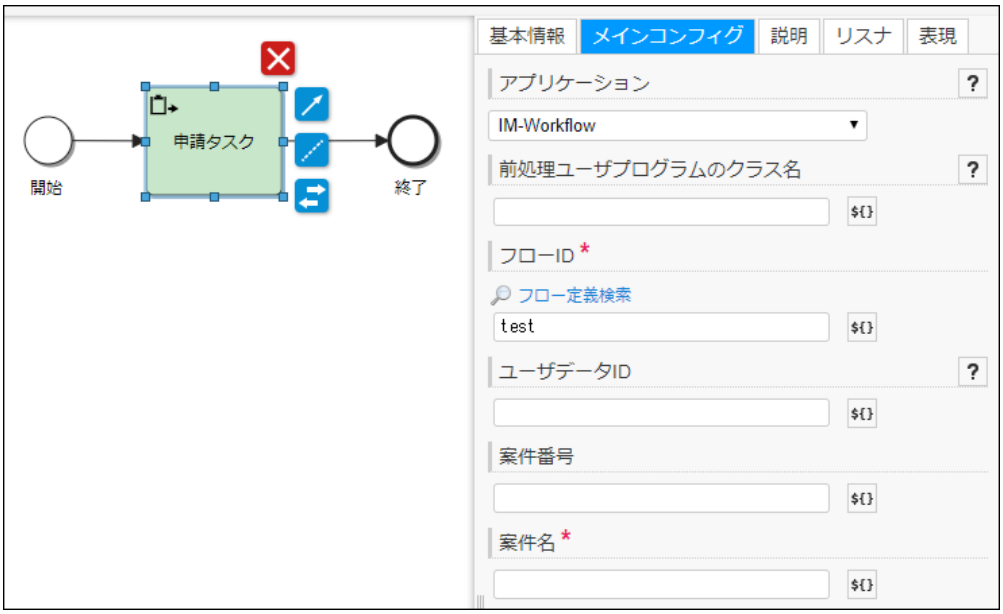
IM-Workflow

IM-BPMから、IM-Workflowと連携する方法を説明します。

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：申請タスク

> **コラム**
>
> 前処理ユーザプログラムの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「申請タスク」を参照してください。

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

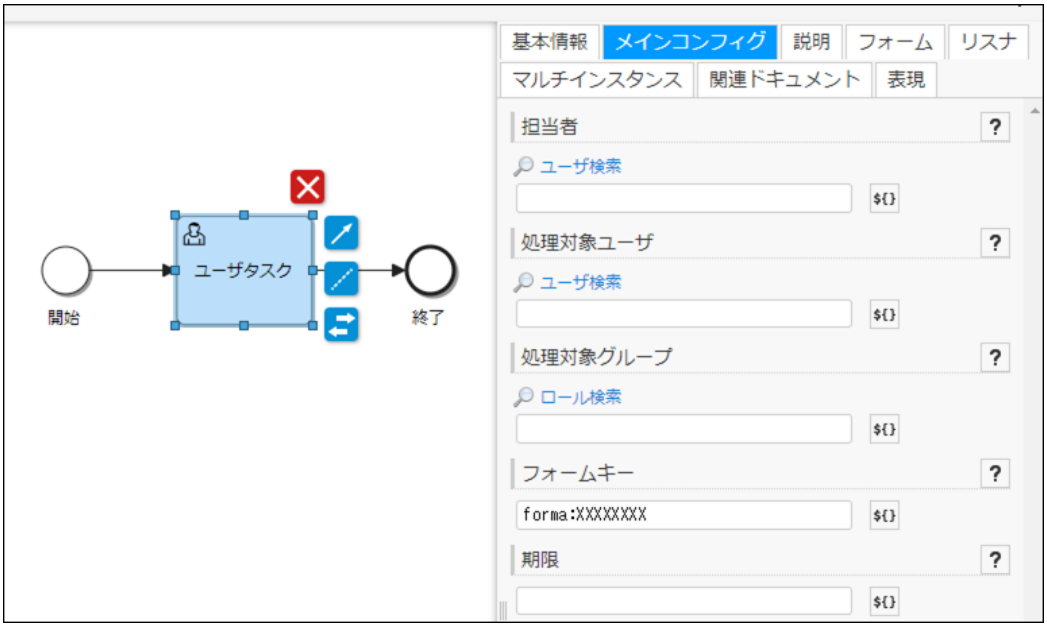
- 前処理、後処理をカスタマイズする
- 起票・申請タスクに前処理ユーザプログラムを設定する

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」を参照してください。

IM-BPMのユーザタスクの フォームキーにIM-FormaDesignerのIDを指定することでIM-FormaDesignerと連携できます。



図：フォームキー

> **コラム**
>
> ユーザタスクの フォームキー についての詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「ユーザタスク」を参照してください。

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。

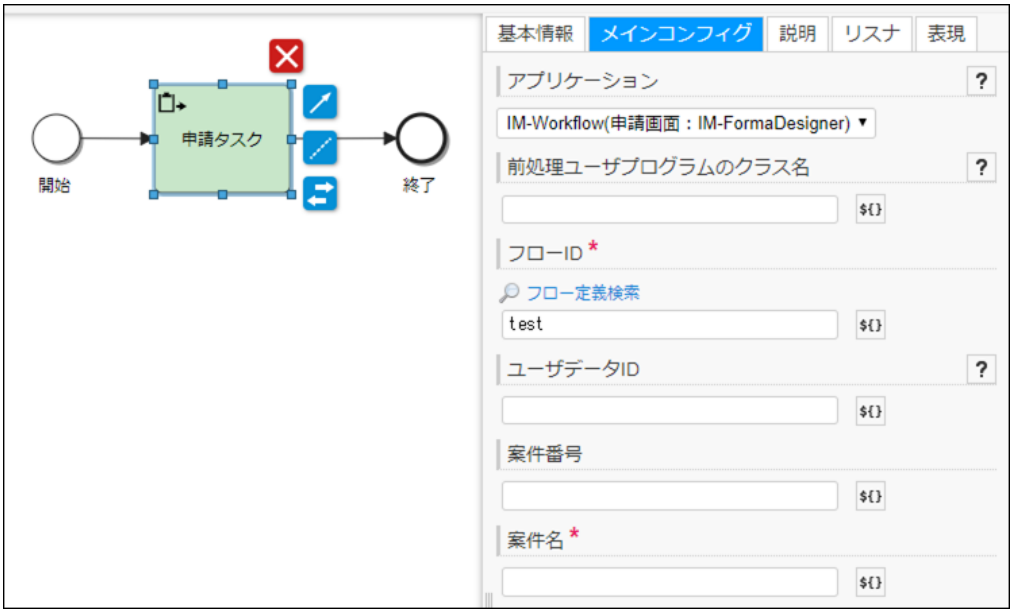
- 前処理 - jp.co.intra_mart.activiti.extension.forma.BPMPreProcessExecutor
- 後処理 - jp.co.intra_mart.activiti.extension.forma.BPMPostProcessExecutor

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：申請タスク

> **コラム**
>
> 前処理ユーザプログラムの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「申請タスク」を参照してください。

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：申請タスク

> **コラム**
>
> 前処理ユーザプログラムの設定の詳細は「IM-BPM プロセスデザイナ 操作ガイド」 - 「申請タスク」を参照してください。

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);
   }
}
```