intra-mart WebPlatform/AppFramework Ver.7.0

BPM プログラミングガイド

2008/08/22 初版

<< 変更履歴 >>

変更年月日	変更内容
2008/08/22	初版

<< 目次 >>

I		1
1.1 は	じめに	1
2 リレーシ	/ョン・フォーム	2
2.1 は	じめに	2
2.1.1	概要	2
2.2 Jav	vaEE開発モデル	3
2.2.1	モデルの作成	3
2.2.2	画面の作成 (JavaEE開発)	4
2.3 ス:	クリプト開発モデル	
2.3.1	モデルの作成	14
2.3.2	画面の作成(スクリプト開発)	14
3 進捗通	知拡張	
	じめに	
	提条件	
	步一覧画面 拨一覧画面	
	步 	
	ントワークフロー連携	
	じめに	
	vaEE開発モデル	
4.2.1	Webサービスの作成	
4.2.2	Webサービスのデプロイ	
4.2.3	ビジネス・プロセス・ダイアグラムの作成	
4.2.4	後処理プログラムの作成	
	クリプト開発モデル	
4.3.1	Webサービスの作成	
4.3.2	Webサービスのデプロイ	
4.3.3	ビジネス・プロセス・ダイアグラムの作成	
4.3.4	WSDLファイルの配置	
4.3.5	後処理プログラムの作成	
	じめに	
	oticeProgressManager	
5.2.1	プロセス詳細情報の登録	
5.2.2	進捗通知タスク情報の登録/更新、処理ユーザ情報の登録	
5.2.3	進捗通知タスクステータスの更新、処理ユーザ情報の登録	
5.2.4	世沙田 カクヘンヘン 一クへの 史材、 処 生 ユー リ 旧 報 い 登 球	
5.2.5	プロセス詳細情報の削除	
5.2.6	プロセス一覧の取得	
5.2.7	プロセス件数の取得	
	skManager	
5.3.1	タスクの取得	
5.3.2	タスク一覧の取得	
5.3.3	タスク件数の取得	
5.3.4	起票タスク一覧の取得	
5.3.5	処理タスク一覧の取得	
5.3.6	通知タスク一覧の取得	74

5.3.7	処理済タスク一覧の取得	75
5.3.8	プロセスの開始	75
5.3.9	処理タスクの完了	76
5.3.10	処理タスクの保留	76
5.3.11	処理タスクの保留→完了	77
5.3.12	処理タスクの保留取消	
5.3.13	処理タスクの一時保存	
5.3.14	通知タスクの確認	
5.4 Tol	kenManager	79
5.4.1	トークンの生成	79
5.4.2	ユーザIDの変換	79
5.4.3	ロール一覧の取得	80
5.4.4	トークンの解析	80

1 概要

1.1 はじめに

本ドキュメントではim-BPMとiWP/iAFを連携、あるいは機能の拡張に関わるプログラミングについて解説します。 概要は以下の通りです。

リレーション・フォーム

ヒューマンタスク用の画面を、スクリプト開発モデルあるいは JavaEE 開発モデルで作成する方法について解説します。作成した画面と BPM フローは、BPM|Designer でリレーション・フォームを作成して連携します。

進捗通知拡張

進捗通知画面をカスタマイズして、独自データを表示する方法について解説 します。

ドキュメントワークフロー連携

BPM フローからドキュメントワークフローを起票するための WebService の作成 方法と、BPM へ処理を戻すための後処理コーディングについて解説します。

API 使用例

im-BPM 用 API のサンプルコードです。

API については API リストの以下の項目も併せて参照してください。

- ・スクリプト開発モデル BPM API
- ·JavaEE 開発モデル BPM API

2 リレーション・フォーム

2.1 はじめに

intra-mart リレーションフォームを介した Intra-mart 画面との連携方法について説明します。

※ リレーション・フォームの作成方法については、「BPM|Designer 操作ガイド 2.2.2 リレーション・フォーム の作成」を参照ください。

※ BPM|Server への処理は、TaskManager の API を利用して行います。TaskManager の詳細は、API リストを参照下さい。

2.1.1 概要

intra-mart リレーションフォームより自動生成されたモデルを使用した2つの開発パターン、JavaEE 開発モデルとスクリプト開発モデルを例にとって、以下の処理の実装方法について説明します。

■ データの取得/返却

BPM|Server と連携した Intra-mart の画面を作成する場合、BPM|Server から送受信されるデータが XML データの為、画面側で認識できる入力値または出力値に変換する必要があります。

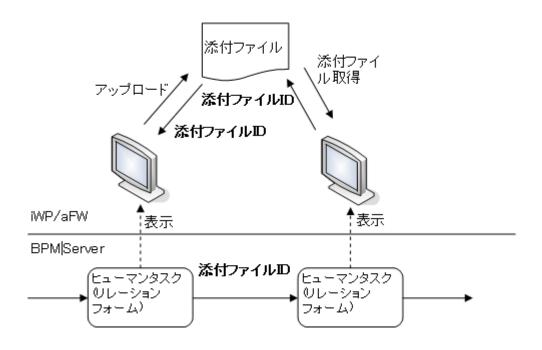
この入出力のXML データは、intra-mart リレーションフォームでモデル生成を行うことにより自動的に入出力のモデルクラスを生成することができます。

データの取得/返却はこのモデルクラスを介して行います。

■ 添付ファイル

BPM で添付ファイルのように、サイズが大きくなりやすいデータを扱う場合、プロセス上にそのままデータを流してしまうことはパフォーマンスの観点から良くありません。

このような場合は、ファイルの実体は iWP/iAF で管理して、BPM プロセス上ではそのファイルを一意に特定する ID(添付ファイル ID)のみを制御する実装がベターと言えます。



2.2 JavaEE開発モデル

javaEE 開発モデルを用いた実装例を説明します。

2.2.1 モデルの作成

JavaEE 開発モデル用のモデル生成を行います。 モデル生成の手順については、「BPM|Designer 操作ガイド 2.2.4 モデル生成」を参照ください。 ここでは以下のモデルを作成します。

- データ取得 ・・・ 入力値モデル(AcknowledgeRequest.java)
- データ返却 ・・・ 出力値モデル(AcknowledgeResponse.java)

自動生成時にファクトリークラス(ObjectFactory.java)も同時に生成されますが、通常使用する必要はありません。

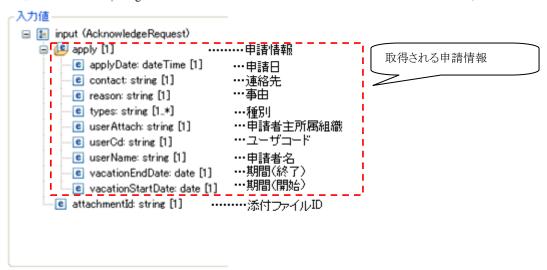
2.2.2 画面の作成(JavaEE開発)

ここでは以下の実装例について説明します。

- データの取得
- データの返却
- 添付ファイル

2.2.2.1 データの取得

自動生成された入力値モデル(AcknowledgeRequest.java)を使用した、データの取得について。 入力値モデルは BPM|Designer のフォームリレーションエディタで以下のように設定してあります。



BPM|Server からデータの取得を行うには以下の実装が必要です。

■ 画面(JSP) : 申請情報を表示します。■ HelperBean : 申請情報を取得します。

apply(申請情報)を BPM|Server から取得する場合、HelperBean では以下のような処理を行いデータを取得します。

- ① リクエストよりタスク ID、実行ユーザを取得。
- ② タスク ID、実行ユーザを元にタスク情報を取得。
- ③ タスク情報より入力値 XML を取得。(入力値モデルの定義に準じた XML 情報)
- ④ 入力値 XML を入力値モデル (AcknowledgeRequest) へ変換。
- ⑤ 入力値モデル(AcknowledgeRequest)より申請情報を取得、変数 apply(申請情報)を生成し格納。

取得された変数 apply(申請情報)は、画面(JSP)の各項目に以下のように反映します。

■ 画面(JSP)

データ取得の為、HelperBean を id="bean"で定義します。

<imfw:HelperBean id="bean" class="sample.bpms.vacation.helperbean.ActivityHelperBean" />

id="bean"で定義された HelperBean の変数 apply(申請情報)から各項目の値を表示します。

```
<div id="applyDiv">
 <!-- 申請者 -->
   >
     申請者
     <c:out value="${bean.apply.userName}" />
   <!-- 申請者主所属組織 -->
   >
     所属
     <c:out value="$[bean.apply.userAttach]" />
   <!-- 申請日 -->
     申請日
     <fmt:formatDate pattern="yyyy/MM/dd" value="${bean.applyDate}" />
     \langle td \rangle
   <!-- 期間 -->
   >
     期間
     <fmt:formatDate pattern="yyyy/MM/dd" value="${bean.vacationStartDate}" />&nbsp;-&nbsp;
        <fmt:formatDate pattern="yyyy/MM/dd" value="${bean.vacationEndDate}" />
     <!-- 種別 -->
   \langle tr \rangle
     種別
     <c:forEach items="${bean.apply.types}" var="type">
          <c:out value="${type}" />&nbsp;
        </c:forEach>
     <!-- 事由 -->
   \langle tr \rangle
     事由
      <div class="textarea">
              value="${fn:replace(fn:escapeXml(bean.apply.reason), bean.lineSeparator,
                                                     '<br
                                                         />')}'
escapeXml="false" />
        </div>
      <!-- 連絡先 -->
     連絡先
      <div class="textarea">
         />')}"
escapeXml="false" />
        </div>
      \langle /div \rangle
```

HelperBean では、TaskManager の API を利用して BPM|Server からデータの取得を行います。

■ HelperBean

```
リクエストよりタスク ID、実行ユーザを取得します。
```

(TaskManager インスタンス生成時に使用します)

```
// リクエスト情報を格納します。
HttpServletRequest request = getRequest();

// タスク ID
String taskId = request.getParameter("taskId");

// タスク ID を格納します
setTaskId(taskId);

// 実行ユーザ
String runUser = request.getParameter("runUser");
```

タスク ID、実行ユーザを元にタスク情報を取得します。

TaskManager の引数について

- ◆ 第一引数: ユーザ ID(リクエストパラメータより取得)
- ◆ 第二引数: ログイングループ ID(セッション情報から取得)

タスク情報より入力値 XML を取得します。

(この処理は未処理タスクなので ActivityTask にキャストしますが、HelperBean の場合は InitialTask、通知タスクの場合は NotificationTask にキャストします)

入力値 XML を入力値モデル (AcknowledgeRequest) へ変換します。

入力値モデル (AcknowledgeRequest)より申請情報を取得、変数 apply (申請情報)を生成し格納します。 // 申請情報を設定します。

setApply(acknowledgeRequest.getApply());

protected XMLGregorianCalendar vacationStartDate;

apply(申請情報)には入力値モデルで設定した各項目の情報が設定されます。 public static class Apply {

@XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected XMLGregorianCalendar applyDate; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected String contact; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected String reason; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected List<String> types; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected String userAttach; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected String userCd; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected String userName; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true) protected XMLGregorianCalendar vacationEndDate; @XmlElement(namespace = "http://javaee.sample.intra-mart.co.jp/forms/activity/imform", required = true)

2.2.2.2 データの返却

自動生成された出力値モデル(AcknowledgeResponse.java)を使用した、データの返却について。 出力値モデルは以下のように設定してあります。



BPM|Server ヘデータの返却を行うには以下の実装が必要です。

■ 画面(JSP) : 承認情報の入力を行います。

■ ControllerObject: ServiceController へ受渡すパラメータクラス。

■ ServiceController : 完了処理を行います。

acknowledge (承認情報)を BPM|Server へ返却する場合、ServiceController で以下のような処理を行いデータを返却します。

- ① ControllerObject のリクエスト情報を取得。
- ② 出力値モデル (AcknowledgeResponse) のインスタンス化とデータの設定。
- ③ 出力値モデルを XML に変換。
- ④ TaskManager に XML を渡し、完了処理を実行。

画面で入力されたデータが BPM|Server へ渡され完了処理が実行されたことにより、BPM|Server の処理フローで 処理待ちになっていたプロセスが再開します。

画面(JSP)では acknowledge (承認情報)の各項目値を入力します。

■ 画面(JSP)

画面より入力される承認情報。

```
<!-- 承認 -->
     \langle tr \rangle
        承認
         <c:if test="${bean.acknowledge.approve}">
               <input type="radio" id="approve_true" name="approve" value="true" checked="checked" />
                <label for="approve_true">承認</label>
                <input type="radio" id="approve_false" name="approve" value="false" />
                <label for="approve_false">否認</label>
            </c:if>
            <c:if test="${not bean.acknowledge.approve}">
                <input type="radio" id="approve_true" name="approve" value="true" />
                <label for="approve_true">承認</label>
                <input type="radio" id="approve_false" name="approve" value="false" checked="checked" />
                <label for="approve_false">否認</label>
            </c:if>
         <!-- コメント -->
     >
        コメント
         <textarea name="comment" class="default" rows="4" cols="35">
               <c:out value="${bean.acknowledge.comment}" />
            </textarea>
         </div>
```

画面で設定された acknowledge(承認情報)は、ControllerObject に格納され ServiceController へ渡されます。

■ ControllerObject

画面(JSP)から渡されるリクエスト情報を定義します。

```
/** タスクID */
private String taskId = null;

/** 実行ユーザ */
private String runUser = null;

/** 承認 */
private String approve = null;

/** コメント */
private String comment = null;
```

ServiceController では TaskMnager を使用して完了処理を行います。

ServiceController

ControllerObject のリクエスト情報を取得します。

 $Activity Complete Controller Object \ form = (Activity Complete Controller Object) \ get Controller Object \ (); \\$

```
出力値モデル (AcknowledgeResponse) のインスタンス化とデータの設定を行います。
// 出力値インスタンスを生成します。
// sample.bpms.vacation.model.AcknowledgeResponse クラスは
// BPMS Designer 上で生成されたクラスです。
AcknowledgeResponse acknowledgeResponse = new AcknowledgeResponse();
// 一時保存する内容を格納します。
AcknowledgeResponse.Acknowledge acknowledge = new AcknowledgeResponse.Acknowledge();
acknowledge Response. set Acknowledge (acknowledge); \\
acknowledge.setUserCd(form.getRunUser());
acknowledge. \\ \textbf{setUserName} (UserHelper.getUserName (form.getRunUser(), not better a set of the property o
              {\tt getUserInfo().getLoginGroupID()},\ {\tt getUserInfo().getLocale()},
              acknowledgeDate.getTime()));
acknowledge. \\ \textbf{setUserAttach} (UserHelper.getUserAttach (form.getRunUser(), \\
              getUserInfo().getLoginGroupID(), getUserInfo().getLocale(),
              acknowledgeDate.getTime()));
acknowledge. \textbf{set} \textbf{Acknowledge} \textbf{Date} (Date Helper
              .toXMLGregorianCalendar(acknowledgeDate));
acknowledge.setComment(form.getComment());
acknowledge. \textbf{setApprove} (Boolean.parseBoolean (form.getApprove())); \\
出力値モデルを XML に変換します。
// 出力値 XML に変換を行います。
Document outputXML = null;
try {
    outputXML = XMLConversionUtils
                        .convertToDocument(acknowledgeResponse);
} catch (XMLConversionException e) {
    throw new ApplicationException("出力値の変換に失敗しました。", e);
TaskManager に XML を渡し、完了処理を実行します。
// タスクマネージャインスタンスを生成します。
TaskManager taskManager = new TaskManager(form.getRunUser(),
              getUserInfo().getLoginGroupID());
// 完了を行います。
try {
    task Manager. {\color{red} \textbf{claimAndCompleteTask}} (form.getTaskId(),\ outputXML);
} catch (BPMSApplicationException e) {
    throw new ApplicationException("完了に失敗しました。", e);
```

完了処理が実行されたことにより、BPM|Serverの処理フローで処理待ちになっていたプロセスが再開します。

2.2.2.3 添付ファイル

添付ファイルは、添付ファイル ID, ログイングループ ID を使用し一意に生成されたフォルダで管理します。 添付ファイル ID は入力値モデルを介して BPM|Server より取得します。 添付ファイルの登録、参照、削除を行う処理の実装方法について説明します。

■ 登録

添付ファイル ID、ログイングループ ID を元にフォルダを特定し、ファイルを保存します。

リクエストより添付ファイル ID とファイル情報を取得し保存ファイル名を生成します。

```
// 添付ファイル ID
final String attachmentId = getRequest().getParameter("attachmentId");

// アップロードファイル
final Entity fileEntity = getEntity("file");

// アップロードされたファイル名
String fileName = fileEntity.getFileName();

// 別名
final String name = getRequest().getParameter("name");

// 別名が存在する場合、ファイル名を上書きします。
if (!"".equals(name)) {
    fileName = name;
}
```

添付ファイル ID、ログイングループ IDを元にフォルダインスタンスを生成します。

添付ファイルを保存します。

```
// アップロードされた内容の保存を行います。

// ここでアップロードされた内容がメモリ上に展開されてしまう為、

// 巨大なファイルサイズを扱う場合 OutOfMemory が発生する可能性があります。

if (!file.save(fileEntity.getBytes())) {

// ファイルの保存に失敗した場合

throw new ApplicationException(fileName + "の登録に失敗しました。");

}。
```

■ 参照(ファイル一覧取得)

添付ファイル ID、ログイングループ ID でフォルダを特定し、中のファイル情報を取得します。

リクエストの添付ファイル ID とログイングループ ID を元にフォルダインスタンスを生成。

フォルダが存在すれば添付ファイル一覧情報を取得。

```
// フォルダが存在した場合、ファイル数をグローバス変数にセットします。
if (folder.exists()) {
    setAttachmentFiles(folder.files());
    setAttachmentFileCount(getAttachmentFiles().size());
}
```

■ 削除

添付ファイル ID、ログイングループ ID、ファイル名を指定しファイルを削除します。

添付ファイル ID、ログイングループ ID、ファイル名より、削除ファイルインスタンスを生成します。

ファイルの削除を行います。

```
if (!<mark>file.remove</mark>()) {

// ファイルの削除に失敗した場合

throw new ApplicationException(attachmentFileName

+ "の削除に失敗しました。");
}
```

添付フォルダ内にファイルが存在しない場合、フォルダを削除します。

```
// 添付フォルダインスタンスを生成します。
NetworkFile folder = new NetworkFile(folderPath);

// 添付フォルダ内にファイル、フォルダが存在しない場合、添付フォルダを削除します。
if (folder.lists().size() == 0) {
    folder.remove();
}
```

2.3 スクリプト開発モデル

スクリプト開発モデルを用いた実装例を説明します。

2.3.1 モデルの作成

スクリプト開発モデル用のモデル生成を行います。

モデル生成の手順については、「BPM|Designer 操作ガイド 2.2.4 モデル生成」を参照ください。 ここでは以下のモデルを作成します。

- データ取得 ・・・ 入力値モデル(AcknowledgeRequest.js)
- データ返却 ・・・ 出力値モデル(AcknowledgeResponse.js)

ファイル名の設定はリレーションフォームで行って下さい。

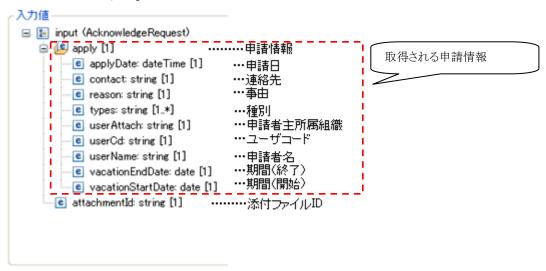
2.3.2 画面の作成(スクリプト開発)

ここでは以下の実装例について説明します。

- データの取得
- データの返却
- 添付ファイル

2.3.2.1 データの取得

自動生成された入力値モデル(AcknowledgeRequest.js)を使用した、データの取得について。 入力値モデルは BPM|Designer のフォームリレーションエディタで以下のように設定してあります。



BPM|Server からデータの取得を行うには以下の実装が必要です。

- 画面(HTML): 申請情報を表示します。
- スクリプト(JavaScript) : 申請情報を取得します。

apply(申請情報)を BPM|Server から取得する場合、スクリプトの初期処理で以下のような処理を行いデータを取得します。

- ① リクエストよりタスク ID、実行ユーザを取得。
- ② タスク ID、実行ユーザを元にタスク情報を取得。
- ③ タスクより入力値 XML を取得し、入力値モデル (acknowledgeRequest) へ変換。
- ④ 入力値モデル (acknowledgeRequest)より申請情報を取得。
- ⑤ 申請情報をグローバル変数\$data の apply(申請情報)を生成し格納。

取得された変数 apply(申請情報)は、画面(HTML)の各項目に以下のように反映されます。

■ 画面(HTML)

データ取得の為、スクリプトの\$data の変数 apply(申請情報)から各項目の値を表示します。

```
<DIV id="applyDiv">
        <TABLE class="topMargin detail table_border_bg" align="center">
                        <!-- 申請者 -->
                       <TR>
                                       <TD class="list_title_bg_left" nowrap="nowrap">申請者</TD>
                                       <TD class="list_data_bg_left"><IMART type="string" value=$data.apply.userName></IMART></TD>
                        <!-- 申請者主所属組織 -->
                        <TR>
                                       <TD class="list_title_bg_left" nowrap="nowrap">所属</TD>
                                       \verb|\TD class="list_data_bg_left"> < IMART type="string" value= $data.apply.userAttach> < / IMART > < / TD> | IMART | | IMART 
                        <!-- 申請日 -->
                       <TR>
                                       <TD class="list_title_bg_left" nowrap="nowrap">申請日</TD>
                                       <TD class="list_data_bg_left">
                                                       <IMART type="date" format="yyyy/MM/dd" value=$data.apply.applyDate></IMART>
                                        </TD>
                        </TR>
                        <!-- 期間 -->
                        <TR>
                                       <TD class="list_title_bg_left" nowrap="nowrap">期間</TD>
                                        <TD class="list_data_bg_left">
                                                      \verb| <IMART type="string" value= $ data. \verb| apply.vacation StartDate| > </IMART > & nbsp; - & nb
                                                       <IMART type="string" value=$data.apply.vacationEndDate>
                                        </TD>
                        </TR>
                       <!-- 種別 -->
                        <TR>
                                        <TD class="list_title_bg_left" nowrap="nowrap">種別</TD>
                                       <TD class="list_data_bg_left">
                                                       <IMART type="repeat" list=$data.apply.types item="$type">
                                                                       <IMART type="string" value=$type></IMART>&nbsp;
                                                       </IMART>
                                      </TD>
                        </TR>
                        <!-- 事由 -->
                        <TR>
                                       <TD class="list_title_bg_left" nowrap="nowrap">事由</TD>
                                           <TD class="list_data_bg_left">
                                                      <DIV class="textarea"><IMART type="string" value=$data.apply.reason></IMART></DIV>
                                           </TD>
                        </TR>
                        <!-- 連絡先 -->
                        <TR>
                                        <TD class="list_title_bg_left" nowrap="nowrap">連絡先</TD>
                                           <TD class="list_data_bg_left">
                                                       <DIV class="textarea"><IMART type="string" value=$data.apply.contact></IMART></DIV>
                                           </TD>
                        </TR>
        </TABLE>
</DIV>
```

スクリプトでは、TaskManager の API を利用して BPM|Server からデータの取得を行います。

■ スクリプト(JavaScript)

リクエストよりタスク ID、実行ユーザを取得します。

(TaskManager インスタンス生成時に使用します)

```
// タスク ID をグローバル変数へ格納

$taskId = request.taskId;

// 実行ユーザをグローバル変数へ格納

$runUser = request.runUser;
```

タスク ID、実行ユーザを元にタスク情報を取得します。

TaskManager の引数について

- ◆ 第一引数: ユーザ ID(リクエストパラメータより取得)
- ◆ 第二引数: ログイングループ ID(セッション情報から取得)

```
// ユーザ情報を取得します。
```

var sessionInfo = AccessSecurityManager.getSessionInfo();

```
// タスク情報を取得します。
```

// TaskManager インスタンスを生成します。

var taskManager = new TaskManager(\$runUser, sessionInfo.loginGroup);

// タスク情報の取得を行います。

var result = taskManager.getTask(\$taskId);

// 取得結果の確認を行います。

if(result.error){

// タスクの取得に失敗

Module.alert.write("SYSTEM.ERR", "タスクの取得に失敗しました。");

}else if(result.data == null){

// タスクが存在しない場合

Module.alert.write("SYSTEM.DATA.NOTHING", "タスクが存在しません。");

}

// タスク情報の格納

var task = result.data;

タスクより入力値 XML を取得し、入力値モデル (acknowledgeRequest) へ格納します。

/*

- * BPMS Designer(IMForm)から生成された AcknowledgeRequest クラス, AcknowledgeResponse クラスを取得します。
- * 生成されたクラスの内容は下記のファイルを参照してください。
- * sample/bpms/vacation/activity/AcknowledgeResponse.js
- $* \ sample/bpms/vacation/activity/AcknowledgeRequest.js \\$

*/

var requestContent = new Content("sample/bpms/vacation/activity/AcknowledgeRequest");

var AcknowledgeRequest = requestContent.getFunction("AcknowledgeRequest");

var responseContent = new Content("sample/bpms/vacation/activity/AcknowledgeResponse");

var AcknowledgeResponse = responseContent.getFunction("AcknowledgeResponse");

// acknowledgeRequest インスタンスを生成します。

var acknowledgeRequest = new AcknowledgeRequest(task.inputXML);

入力値モデル (acknowledgeRequest) より申請情報を取得します。

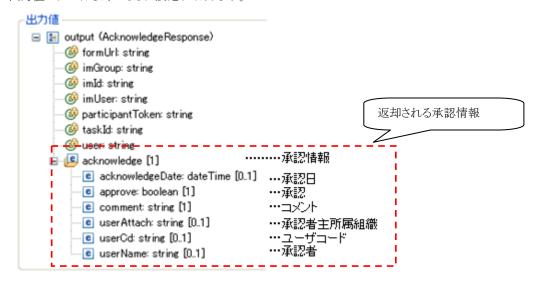
```
// 申請情報を格納
var apply = acknowledgeRequest.getApply();
```

申請情報をグローバル変数\$data の apply(申請情報)へ格納します。

```
// 入力値をグローバル変数へ格納します。
$data = {
     // 添付ファイル ID
     attachment Id: acknowledge Request.get Attachment Id(),\\
     // 申請情報
     apply: {
          userCd: Module.string.browse(apply. {\color{red} {\bf getUserCd}}()), \\
          userName: Module.string.browse(apply. {\color{red} {\bf getUserName}())},
          userAttach: Module.string.browse(apply.getUserAttach()),
          applyDate: apply.getApplyDate(),
          vacation Start Date: Format.from Date ("yyyy/MM/dd", apply. {\tt getVacationStartDate}()), \\
          vacation End Date: Format.from Date ("yyyy/MM/dd", apply. {\tt getVacationEndDate}()), \\
          types: apply.getTypes(),
          reason: Module.string.browse(apply.getReason()),
          contact: Module.string.browse(apply. {\color{red} {\bf getContact}()}), \\
     }
```

2.3.2.2 データの返却

自動生成された出力値モデル(AcknowledgeResponse.js)を使用した、データの返却について。 出力値モデルは以下のように設定してあります。



BPM|Server ヘデータの返却を行うには以下の実装が必要です。

- 画面(HTML): 承認情報の入力を行います。
- スクリプト(JavaScript) : 完了処理を行います。

acknowledge(承認情報)を BPM|Server へ返却する場合、スクリプトで以下のような処理を行いデータを返却します。

- ① リクエスト情報を取得。
- ② 出力値モデル (AcknowledgeResponse) のインスタンス化。
- ③ 出力値モデルにデータの設定を行い、XML へ変換。
- ④ TaskManager に XML を渡し、完了処理を実行。

画面で入力されたデータが BPM|Server へ渡され完了処理が実行されたことにより、BPM|Server の処理フローで 処理待ちになっていたプロセスが再開します。

画面(HTML)では acknowledge(承認情報)の各項目値を入力します。

■ 画面(HTML)

```
画面より入力される承認情報。
```

```
<!-- 承認 -->
<TR>
    <TD class="list_title_bg_left" nowrap="nowrap">
        <SPAN class="bold">承認<FONT class="attention">(必須)</FONT></SPAN>
    </TD>
     <TD class="list_data_bg_left">
      <IMART type="condition" validity=$data.acknowledge.approve>
         <LABEL for="approve_true">承認</LABEL>
         <IMART type="input" id="approve_false" style="radio" name="approve" value="false"></IMART>
      <LABEL for="approve_false">否認</LABEL>
       </IMART>
      <IMART type="condition" validity=$data.acknowledge.approve negative>
        <IMART type="input" id="approve_true" style="radio" name="approve" value="true"></IMART>
        <LABEL for="approve_true">承認</LABEL>
        <IMART type="input" id="approve_false" style="radio" name="approve" value="false" checked="checked">
        </IMART>
        <LABEL for="approve_false">否認</LABEL>
      </IMART>
    </TD>
</TR>
<!-- コメント -->
<TR>
   <TD class="list_title_bg_left" nowrap="nowrap">
      〈SPAN class="bold">コメント〈FONT class="attention"〉(必須)〈/FONT〉〈/SPAN〉
    </TD>
    <TD class="list_data_bg_left">
      <TEXTAREA name="comment" class="default" rows="4" cols="35">
        <IMART type="string" value=$data.acknowledge.comment></IMART>
      </TEXTAREA>
    </TD>
</TR>
```

スクリプトでは TaskMnager を使用して完了処理を行います。

■ スクリプト(JavaScript)

リクエスト情報を取得します。

```
function completeAction(request){

// タスク ID を格納します。
var taskId = request.taskId;

// 実行ユーザを格納します。
var runUser = request.runUser;

// ユーザ情報を取得します。
var sessionInfo = AccessSecurityManager.getSessionInfo();

// 承認内容を格納します。
var approve = request.approve == "true";

// コメントを格納します。
var comment = request.comment;
```

出力値モデル(AcknowledgeResponse)のインスタンス化を行います。

出力値モデルにデータの設定を行い、XML へ変換します。

```
// 完了を行う内容をセットします。
acknowledge.setApprove(approve);
acknowledge.setComment(comment);
acknowledge.setAcknowledgeDate(acknowledgeDate);
acknowledge.setUserCd(runUser);
acknowledge.setUserName(userName);
acknowledge.setUserAttach(userAttach);
acknowledgeResponse.setAcknowledge(acknowledge);

// 完了内容を XML 型に変換します。
var outputXML = acknowledgeResponse.toXML();
```

TaskManager に XML を渡し、完了処理を実行します。

```
// タスクマネージャインスタンスを生成します。
var taskManager = new TaskManager(runUser, sessionInfo.loginGroup);
// 保留+完了を行います。
var result = taskManager.claimAndComplete(taskId, outputXML);
// 完了結果の確認を行います。
if(result.error){
    // 完了に失敗した場合
    // 承認画面へ遷移します。
    forward(Web.current(), {
       taskld: taskld.
       runUser: runUser,
       referer: request.referer,
       errorMessage: "完了に失敗しました。"
   });
}else{
    // 完了に成功した場合
    // 一覧画面へ遷移します。
   forward(request.referer, {});
```

完了処理が実行されたことにより、BPM|Serverの処理フローで処理待ちになっていたプロセスが再開します。

2.3.2.3 添付ファイル

添付ファイルは、添付ファイル ID, ログイングループ ID を使用し一意に生成されたフォルダで管理します。 添付ファイル ID は入力値モデルを介して BPM|Server より取得します。

添付ファイルの登録、参照、削除を行うスクリプトの実装方法について説明します。

■ 登録

添付ファイル ID、ログイングループ ID を元にフォルダを特定し、ファイルを保存します。

リクエストより添付ファイル ID とファイル情報を取得し、保存ファイル名を生成します。

```
// 添付ファイル ID
var attachmentId = request.attachmentId;

// アップロードファイル
var fileParameter = request.getParameter("file");

// アップロードされたファイル名
var fileName = fileParameter.getFileName();

// 別名
var name = request.name;

// 別名が存在する場合、ファイル名を上書きします。
if(name != ""){
    fileName = name;
}
```

セッションよりログイングループ ID を取得し、フォルダインスタンスを生成します。

```
// セッション情報を取得します。
var sessionInfo = AccessSecurityManager.getSessionInfo();

// 添付フォルダパスを生成します。
var folderPath = createFolderPath(attachmentId, sessionInfo.loginGroup);

// 添付フォルダインスタンスを生成します。
var folder = new VirtualFile(folderPath);
```

ファイルインスタンスを生成し、ファイルを保存します。

■ 参照(ファイル一覧取得)

添付ファイル ID、ログイングループ ID を指定し、フォルダ中のファイル情報を取得します。

添付ファイル ID、ログイングループ IDを元にフォルダインスタンスを生成します。

フォルダが存在すれば添付ファイル一覧情報を取得。

```
// ファイル一覧格納用
var files = □;

// フォルダが存在するか確認を行います。
if(folder.exist()){
    // フォルダが存在する場合ファイル一覧を取得します。
    files = folder.files();
    // グローバル変数にファイル数を格納します。
    $attachmentFileCount = files.length;
}
```

ファイル名のエスケープ、変換を行い、添付ファイル一覧に格納します。

```
// HTML 側で利用する為にファイル名のエスケープ、変換を行います。
// ファイル数分ループ
for(var i = 0, size = files.length; i < size; i++){
  var file = files[i];
  // エスケープしたファイル名を格納します。
  $attachmentFiles.push({
    name: Module.string.browse(file, " ", "\inftyn"),
    escapeName: Module.string.browse(file)
  });
}
```

■ 削除

添付ファイル ID、ログイングループ ID、ファイル名を指定しファイルを削除します。

```
添付ファイル ID、ログイングループ ID、ファイル名より、削除ファイルインスタンスを生成します。
// 添付ファイル ID
var attachmentId = request.attachmentId;
// 添付ファイル名
var attachmentFileName = request.attachmentFileName;
// セッション情報を取得します。
var sessionInfo = AccessSecurityManager.getSessionInfo();
// 添付フォルダパスを生成します。
var folderPath = createFolderPath(attachmentId, sessionInfo.loginGroup);
// ファイルインスタンスを生成します。
var file = new VirtualFile(folderPath + "/" + attachmentFileName);
 ファイルの削除を行います。
// ファイルの削除を行います。
if(!file.remove()){
   // ファイルの削除に失敗した場合
   Module.alert.reload("SYSTEM.ERR", attachmentFileName + "の削除に失敗しました。", request);
 添付フォルダ内にファイルが存在しない場合、フォルダを削除します。
// 添付フォルダインスタンスを生成します。
var folder = new VirtualFile(folderPath);
// 添付フォルダ内にファイル、フォルダが存在しない場合、添付フォルダを削除します。
if(folder.lists().length == 0){
   folder.remove();
```

3 進捗通知拡張

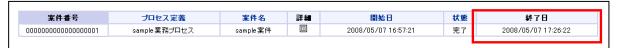
3.1 はじめに

進捗一覧画面と進捗一覧の詳細画面に表示される一覧に、ユーザ固有のデータを表示することができます。ここでは、データを表示するのに必要な設定とプログラミングの例について説明します。

進捗一覧画面

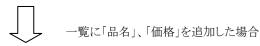


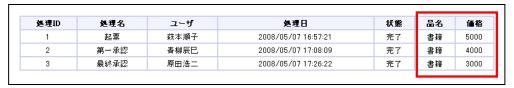




詳細画面







3.2 前提条件

以下の SQL をデータベースに発行してください。

- %StorageService%/storage/bpms/sample_b_bpm_t_purchase.sql
- $\cdot \ \ \%StorageService\%/storage/bpms/sample_notice_progress_insert_data.sql$

3.3 進捗一覧画面

- (1) 設定
 - i) パラメータの変更

進捗一覧設定ファイル([progress.ini]ファイル)のパラメータを変更します。

● 進捗一覧設定ファイル

% ResourceService % /pages/platform/src/bpms/view/progress/progress.ini

進捗一覧設定ファイルのパラメータ[EXTENSION FLG]を「true」に変更します。

≪進捗一覧設定ファイル≫

業務データ追加フラグ

EXTENSION FLG=true

ii) キーの設定

進捗一覧設定ファイル([progress.ini]ファイル)に追加データ用のキーを設定します。

● 進捗一覧設定ファイル

% ResourceService %/pages/platform/src/bpms/view/progress/progress.ini

進捗一覧設定ファイルに追加表示データ用のキーを設定します。設定するキーは、「TABLE COLUMN KEY.連番=任意文字列」という形式で記述します。

≪進捗一覧設定ファイル≫

一覧表示項目

チェックボックス

TABLE_COLUMN_KEY.0=checkbox

案件番号

TABLE_COLUMN_KEY.1=processInstanceId

プロセス定義

TABLE_COLUMN_KEY.2=businessProcessName

案件名

TABLE_COLUMN_KEY.3=processName

詳細

TABLE_COLUMN_KEY.4=processDetail

開始日

TABLE_COLUMN_KEY.5=startDate

状態

TABLE COLUMN KEY.6=processStatus

追加表示データ

TABLE_COLUMN_KEY.7= BUSINESS_DATA_KEY_01

(2) プログラミング

i) 表示データの追加

データ追加処理ファイル ([addBusinessData.js]ファイル) の addProcessHeader メソッド (ヘッダー追加処理) と addProcessBusinessData メソッド (データ追加処理) に、処理を記述します。

● データ追加処理ファイル

% ResourceService %/pages/platform/src/bpms/view/progress/addBusinessData.js

≪データ追加処理ファイルの addProcessHeader メソッド(ヘッダー追加処理)≫

```
// 【機能】追加表示データのヘッダー追加(進捗一覧画面)
// 【 入力 】ヘッダー情報 Array オブジェクト
  【返却值】
  【 概要 】追加表示データ用のヘッダー情報(項目名、表示位置)を追加します。
   【 備考】ヘッダー情報オブジェクト(business data orders)は以下のプロパティをもつ Array 型のオブジェクトです。
          プロパティを以下のように設定してください。
//
             business_data_orders[0..*] (Array)ヘッダー情報オブジェクト
//
                    |-name
                                 (String)項目名
//
                    └align
                                 (String)表示位置 (center:中央、left:左寄せ、right:右寄せ)
function addProcessHeader(business_data_orders){
   // ヘッダー情報
   business_data_orders[0] = new Object();
   business_data_orders[0].name = "終了日"; // 項目名
   business_data_orders[0].align = "center"; // 表示位置
```

≪データ追加処理ファイルの addProcessBusinessData メソッド(データ追加処理)≫

```
【 機能 】追加表示データの追加(進捗一覧画面)
   【入力】検索結果データ
   【返却值】
   【 概要 】検索結果データへ追加表示データを追加します。
   【 備考 】検索結果データ(dispList)は以下のプロパティを持つ Array 型のオブジェクトです。
                                         (Array)検索結果データ
//
               dispList [0..*]
                                         (String)プロセスインスタンス ID
//
                   |-processInstanceId
                   |-businessProcessName
                                         (String)業務プロセス名
//
//
                   |-processName
                                         (String)プロセス名
                   \vdashstartDate
                                         (String)開始日
                                         (String)プロセスステータス
                   |-processStatus
//
                   └noticeTask [0..*]
                                         (Array)進捗通知タスク
                           -processInstanceId
                                             (String)プロセスインスタンス ID
                           -noticeTaskId
                                             (String)タスク ID
                           ├noticeTaskName
                                             (String)タスク名
                           -taskStatus
                                             (String)タスクステータス
                           └userList[0..*]
                                             (Array)処理ユーザ
                                             (String)ユーザコード
                                  ⊢userCd
                                  ∟tranDate
                                             (String)処理日
function addProcessBusinessData(dispList){
    var manager = new ProgressManagerExtension();
    for(var i = 0; i < dispList.length; i++) {
       // プロセスインスタンス ID
       var processInstanceId = dispList[i].processInstanceId;
       // 追加データ取得
       var sql = "SELECT '
                      "MAX(TRAN DATE) AS finish date "
               + "FROM "
                      "B_BPMS_T_USER "
               + "WHERE"
                      "PROCESS_INSTANCE_ID "
               + "IN (SELECT "
                          "PROCESS INSTANCE ID "
                      "FROM "
                          "B BPMS T PROCESS"
                      "WHERE "
                          "PROCESS INSTANCE ID="+ """ + processInstanceId + "" "
                      "AND "
                          "PROCESS_STATUS='2')";
                                      (次ページへ続く)
```

3.4 進捗詳細画面

- (1) 設定
 - i) パラメータの変更

進捗一覧設定ファイル([progress.ini]ファイル)のパラメータを変更します。

● 進捗一覧設定ファイル

% ResourceService %/pages/platform/src/bpms/view/progress/progress.ini

進捗一覧設定ファイルのパラメータ[EXTENSION FLG]を「true」に変更します。

≪進捗一覧設定ファイル≫

業務データ追加フラグ

EXTENSION_FLG=true

ii) キーの設定

進捗一覧設定ファイル([progress.ini]ファイル)に追加データ用のキーを設定します。

● 進捗一覧設定ファイル

% ResourceService % /pages/platform/src/bpms/view/progress/progress.ini

進捗一覧設定ファイルに追加表示データ用のキーを設定します。設定するキーは、 $TASK\ TABLE\ COLUMN\ KEY.連番=任意文字列」という形式で記述します。$

≪進捗一覧設定ファイル≫

一覧表示項目

処理 ID

TASK_TABLE_COLUMN_KEY.0=noticeTaskId

処理名

TASK_TABLE_COLUMN_KEY.1=noticeTaskName

ユーザ

TASK_TABLE_COLUMN_KEY.2=userCd

処理日

 $TASK_TABLE_COLUMN_KEY.3 = tranDate$

状態

 $\underline{TASK_TABLE_COLUMN_KEY.4} = taskStatus$

追加表示データ1

TASK_TABLE_COLUMN_KEY.5= BUSINESS_DATA_KEY_01

追加表示データ2

TASK TABLE COLUMN KEY.6= BUSINESS DATA KEY 02

(2) プログラミング

iii) 表示データの追加

データ追加処理ファイル ([addBusinessData.js]ファイル)の addTaskHeader メソッド (ヘッダー追加処理)と addTaskBusinessData メソッド (データ追加処理) に、処理を記述します。

● データ追加処理ファイル

% ResourceService %/pages/platform/src/bpms/view/progress/addBusinessData.js

≪データ追加処理ファイルの addTaskHeader メソッド(ヘッダー追加処理)≫

```
【機能】追加表示データのヘッダー追加(詳細画面)
   【 入力 】ヘッダー情報オブジェクト
   【返却值】
   【 概要 】追加表示データ用のヘッダー情報(項目名、表示位置)を追加します。
   【 備考 】 ヘッダー情報オブジェクト(business data orders)は以下のプロパティをもつ Array 型のオブジェクトです。
          プロパティを以下のように設定してください。
//
              business_data_orders[0..*] (Array)ヘッダー情報オブジェクト
//
//
                     ⊢name
                                   (String)項目名
//
                      └align
                                   (String)表示位置 (center:中央、left:左寄せ、right:右寄せ)
function\ addTaskHeader(business\_data\_orders)\{
   // ヘッダー1
   business_data_orders[0] = new Object();
   business_data_orders[0].name = "品名";
                                      // 項目名
   business_data_orders[0].align = "center"; // 表示位置
   // ヘッダー2
   business_data_orders[1] = new Object();
                                      // 項目名
   business_data_orders[1].name = "価格";
   business_data_orders[1].align = "center"; // 表示位置
```

≪データ追加処理ファイルの addTaskBusinessData メソッド(データ追加処理)≫

```
// 【 機能 】追加表示データの追加(詳細画面)
// 【 入力 】検索結果データ
  【返却值】
// 【 概要 】検索結果データへ追加表示データを追加します。
   【 備考 】検索結果データ(dispList)は以下のプロパティを持つ Array 型のオブジェクトです。
                                     (Array)検索結果データ
//
               dispList [0..*]
                                      (String)プロセスインスタンス ID
//
                   -processInstanceId
                   |-noticeTaskId
                                      (String)タスク ID
//
                   -noticeTaskName
                                      (String)タスク名
                                      (String)タスクステータス
//
                   |-taskStatus
                   └userList [0..*]
                                      (Array)処理ユーザ
//
                           -userCd
                                      (String)ユーザコード
                           ∟tranDate
                                      (String)処理日
function addTaskBusinessData(dispList){
   var manager = new ProgressManagerExtension();
   for(var i=0; i<dispList.length; i++) {
       // プロセスインスタンス ID
       var processInstanceId = dispList[i].processInstanceId;
       // タスク ID
       var noticeTaskId = dispList[i].noticeTaskId;
       // 追加データ取得
       var sql = "SELECT"
                      "ITEM NAME,"
                      "ITEM_VALUE"
               + "FROM"
                      "SAMPLE B BPM T PURCHASE"
               + "WHERE"
                      " PROCESS INSTANCE ID="+ """ + processInstanceId + """
               + "AND"
                      " NOTICE_TASK_ID=" + """ + noticeTaskId + """;
       var result = DatabaseManager.select(sql);
       // エラー処理
       if(result.error){
           message = "データの取得に失敗しました。";
           return;
                                     (次ページへ続く)
```

```
(前ページの続き)
       // 品名
       var itemName = result.data[0].item_name;
       // 価格
       var itemValue = result.data[0].item_value;
       // データ追加
       manager.set Task Business Data (dispList,\\
                                   processInstanceId,
                                   noticeTaskId,
                                   "BUSINESS_DATA_KEY_01",
                                   itemName);
       manager.set Task Business Data (dispList,\\
                                   processInstanceId,
                                   noticeTaskId,
                                   "BUSINESS_DATA_KEY_02",
                                   itemValue);
}
```

4 ドキュメントワークフロー連携

4.1 はじめに

ここでは、ドキュメントワークフローとの連携の為の Web サービスと Web サービス・クライアントの作成方法について以下のサンプルを例にして説明します。

- ワークフロー連携サンプル(JavaEE 開発モデル)
 - ◆ sample im workflow javaee
- ワークフロー連携サンプル(スクリプト開発モデル)
 - ◆ sample im workflow script

im-BPM とドキュメントワークフローを連携するには、以下の作業が必要になります。

- 連携するドキュメントワークフローのプロセス定義を作成する。
- ② ドキュメントワークフロー開始用のWebサービスの作成する。
- ③ ②で作成した Web サービスを iWP/iAF の ApplicationRuntime ヘデプロイする。
- ④ BPM|Designer より連携用のダイアグラムを作成する。
- ⑤ ドキュメントワークフローの案件終了時に処理結果を BPM|Server へ通知する後処理プログラムを作成する。
- ⑥ ⑤で作成した後処理プログラムを①で作成したプロセス定義に設定する。
 - ※ ①および⑥については本書では説明を行いません。
 - ※ ①および⑥の設定方法については「ワークフロー操作ガイド 2.1.3 プロセス定義の作成」を 参照ください。

この章を読み進める上での注意点は以下の通りです。

- Web サービスに関する簡単な知識が必要です。
 - ◆ 本書で扱う「Web サービス」とは、「**SOAP/WSDL ベースの Web サービス**」のことを意味します。ドキュメントワークフローとの連携は Web サービスを使用して行うので、Web サービスに関する知識は必須となります。 Web サービスについては「Web サービス・プログラミングガイド」を参照ください。
- im-JavaEE Framework の知識が必要です。
 - ◆ この章のサンプルプログラム「ワークフロー連携サンプル(JavaEE 開発モデル)」では、Web サービス・ プロバイダから im-JavaEE Framework のサブフレームワークである、イベントフレームワークを利用して 作成されたアプリケーションを呼び出します。 im-JavaEE Framework の詳しい仕様については、 「intra-mart WebPlatform/AppFramework im-JavaEE Framework 仕様書」を参照してください。
- intra-mart ドキュメントワークフローに関する簡単な知識が必要です。
 - ◆ この章では、ドキュメントワークフローとの連携を行う為、ドキュメントワークフローに関する知識が必須となります。特に、ドキュメントワークフローの API や後処理プログラムについての仕様を理解しておく必要があります。ドキュメントワークフローについては、「ワークフロー仕様書」及び「ワークフロー プログラミングガイド」を参照ください。

4.2 JavaEE開発モデル

ここでは、javaEE 開発モデルを用いたワークフロー連携方法をサンプル(sample_im_workflow_javaee)を使用して説明します。sample_im_workflow_javaee については、「BPM イントロダクション 2.1.2 ワークフロー連携サンプルプロジェクト(JavaEE 開発モデル)」を参照ください。

ここでは連携に必要な以下の手順について説明を行います。

- ◆ ドキュメントワークフローを起票する Web サービスを作成する。
- ◆ iWP/iAF の ApplicationRuntime ~ Web サービスをデプロイする。
- ◆ BPM|Designer を使用し、ビジネス・プロセス・ダイアグラムを作成する。
- ◆ ドキュメントワークフローの処理結果を返却する後処理プログラムを作成する。

4.2.1 Webサービスの作成

まずは、ドキュメントワークフローを起票する為の Web サービスを作成します。 サンプルでは以下のクラスが Web サービスの実装クラスとなります。

■Web サービス実装クラスパス

%ApplicationRuntime%/doc/imart/WEB-INF/classes/sample/bpms/im_workflow/SampleBpwDraftService SampleBpwDraftService.java のソースは以下の通りです。

```
package sample.bpms.im workflow;
import java.util.StringTokenizer;
import org.apache.axis2.AxisFault;
import sample.bpms.im_workflow.event.SampleBpwDraftEvent;
import sample.bpms.im workflow.event.SampleBpwDraftEventResult;
import jp.co.intra mart.foundation.web service.auth.WSUserInfo;
import jp.co.intra_mart.framework.base.event.EventManager;
import jp.co.intra mart.framework.base.util.ConfigurationUserInfo;
import sample.bpms.im workflow.model.object.SampleApplicationInfoModel;
import sample.bpms.im workflow.model.object.SampleBpmnRelationInfoModel;
public class SampleBpwDraftService {
   public String draft(WSUserInfo wsUserInfo, SampleBpmnRelationInfoModel relationInfo,
         SampleApplicationInfoModel applicationInfo) throws AxisFault {
      try {
         // ユーザ情報モデル生成
         String[] user = new String[st.countTokens()];
         int idx = 0;
         while(st.hasMoreTokens()) {
            user[idx++] = st.nextToken();
         ConfigurationUserInfo userInfo = new ConfigurationUserInfo();
         userInfo.setLoginGroupID(user[0]);
         userInfo.setUserID(user[1]);
         // イベント生成
         EventManager eventManager = EventManager.getEventManager();
         SampleBpwDraftEvent event = (SampleBpwDraftEvent)eventManager.createEvent(
                "sample.bpms.im_workflow.conf.bpms_im_workflow", "sample_bpw_draft", userInfo);
         // イベントに申請情報を設定
         event.setRelationInfo(relationInfo);
         event.setApplicationInfo(applicationInfo);
         // イベント実行
         SampleBpwDraftEventResult eventResult =
             (SampleBpwDraftEventResult) eventManager.dispatch (event);
         return eventResult.getModel().getProcessCd();
      } catch (Exception e) {
         e.printStackTrace();
         throw AxisFault.makeFault(e);
   }
```

4.2.1.1 引数に指定されている各要素の定義

この Web サービスで使用する要素は以下の通りです。

① WSUserInfo 認証用ユーザ情報モデル

intra-mart 認証用モデルクラスです。Web サービスの認証を行う為に使用されます。 詳しくは、「Web サービス・プログラミングガイド 3.3 認証モジュール」を参照ください。

② SampleBpmnRelationInfoModel サンプル連係情報モデル

im-BPM との連携情報を保持します。この要素内の各子要素にワークフロー連携を行う為に必要な値を保持します。

SampleBpmnRelationInfoModel の子要素は以下の通りになります。

1 1	1	A STATE OF THE STA
correlationSetKey	相関セットキー	im-BPM との連携用のキー
endpoint	エンドポイント	BPM Server のエンドポイント
wsdlPath	WSDL ファイル配置パス	im-BPM の WSDL 格納先のパス
		※JavaEE 開発モデルでは使用しません。
serviceName	サービス名	処理結果を返却する時に呼び出すサービスの名前
		※JavaEE 開発モデルでは使用しません。
processDefCd	プロセス定義コード	起票するドキュメントワークフローのプロセス定義コード
processName	プロセス名	起票するドキュメントワークフローの案件名
userId	ユーザ ID	起票を行うユーザのユーザ ID

相関セットキーは、im-BPM とドキュメントワークフローの関連付けを行う為のキーになります。サンプルでは、im-BPM のプロセスインスタンス ID をキーとしています。

エンドポイントには、ドキュメントワークフロー完了時に処理結果をBPM|Serverへ返却する際に呼び出されるBPM|Serverのサービスのエンドポイントになります。

③ SampleApplicationInfoModel サンプルアプリケーションモデル

サンプルアプリケーションのアプリケーションデータを保持します。

この例では、ドキュメントワークフローのサンプル

[カスタム]JavaEE 開発モデル・ドキュメントワークフロー1」

と連携しているので、このサンプルアプリケーションの起票に必要な情報を保持しています。 このサンプルは、「物品購買」のサンプルアプリケーションとなっていますので、物品購買の為の要素が定義されています。

SampleApplicationInfoModel の子要素は以下の通りになります。

itemName	品名	品名
itemAmount	数量	購入数
itemPrice	金額	購入品の単価
deliveredPlace	納品場所	納品場所
purchasePurpose	購入目的	購入目的
remarks	備考	備考

4.2.1.2 起票ユーザ情報の設定

このサンプルでは、プロセスを開始したユーザを起票ユーザとする為、プロセスの開始ユーザ情報がSampleBpmnRelationInfoModel の userId 要素に受け渡されます。しかし、im-BPM のユーザ情報は「%ログイングループ ID%¥%ユーザ ID%」の形式になっている為、そのままでは使用出来ません。その為、ログイングループ IDとユーザ IDを切り離す必要があります。

実装例

```
// ユーザ情報モデル生成
StringTokenizer st = new StringTokenizer(relationInfo.getUserId(), "¥¥");
String[] user = new String[st.countTokens()];
int idx = 0;
while(st.hasMoreTokens()) {
    user[idx++] = st.nextToken();
}
ConfigurationUserInfo userInfo = new ConfigurationUserInfo();
userInfo.setLoginGroupID(user[0]);
userInfo.setUserID(user[1]);
```

4.2.1.3 起票イベントの呼び出し

SampleBpwDraftService.java では起票用のイベントを呼び出し、実際の起票処理はそのイベントの中で行っています。

イベント内では以下の処理を行っています。

- 指定されたプロセス定義が申請可能かチェックする。
- アプリケーション情報を DB へ登録する。
- im-BPM 連係情報を DB へ登録する。
- 起票時の所属組織情報を取得する。
- ドキュメントワークフローの起票 API を実行する。

イベント内の処理の実装例は以下のソースを参照してください。

■起票イベント実装クラスパス

 $\% Application Runtime \%/doc/imart/WEB-INF/classes/sample/bpms/im_workflow/event/Sample BpwDraft Event Listener.java$

ドキュメントワークフローの起票処理には、ワークフローモジュール API Drafter クラスを使用しています。 詳しくは、「API リスト」および「ワークフロー プログラミングガイド 3.3 API を使用する」を参照ください。

4.2.2 Webサービスのデプロイ

ドキュメントワークフロー起票用の Web サービスの作成が完了したら、

iWP/iAF の ApplicationRuntime 上へ Web サービスをデプロイします。

サンプルでは、「services.xml」を規定のディレクトリに配置する方法でデプロイを行っています。

Web サービスのデプロイ方法については「Web サービス・プログラミングガイド 6.4 Web サービスのデプロイ方法」を参照ください。

■services.xml の配置ディレクトリ

%ApplicationRuntime%/doc/imart/WEB-INF/services/im bpms samples

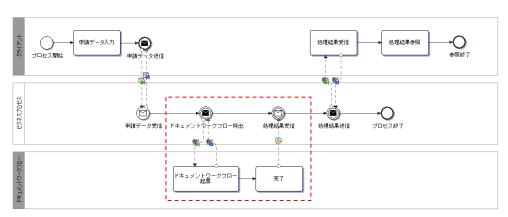
「services.xml」の内容は以下の通りです。

```
6: <service name="im_workflow_javaee" >
     <Description></Description>
8:
g.
      <parameter name="ServiceClass">sample.bpms.im_workflow.SampleBpwDraftService</parameter>
10:
11:
      <module ref="im ws auth"/>
12:
13:
     <messageReceivers>
14:
        <messageReceiver</pre>
                  mep="http://www.w3.org/2004/08/wsdl/in-only"
                  {\tt class=''org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver''} \ / {\tt >}
15:
        <messageReceiver</pre>
                  mep="http://www.w3.org/2004/08/wsdl/in-out"
                  class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
16: </messageReceivers>
17: </service>
```

4.2.3 ビジネス・プロセス・ダイアグラムの作成

サンプルでは、以下のダイアグラムを作成しています。

ワークフロー連携サンプルプロジェクト(JavaEE 開発モデル)については「BPM イントロダクション 2.1.2 ワークフロー連携サンプルプロジェクト(JavaEE 開発モデル)」を参照ください。



点線で囲まれた範囲がワークフローとの連携の部分になります。

4.2.3.1 ワークフロー連携サンプル(Java EE 開発モデル)に含まれるファイル

sample_im_workflow_javaee には以下のファイルが含まれます。

- sample im workflow javaee.bpm
 - ワークフロー連携サンプルのビジネス・プロセス・ダイアグラムです。BPM プロセスの定義等が 設定されています。
- apply.xform
 - 申請用の X フォームです。 申請データを入力し、プロセスを開始する為に使用します。 詳しくは、「BPM|Designer 操作ガイド 1.4.3 XForm を使用したヒューマンタスクの作成手順」を 参照ください。
- ♦ im workflow javaee.wsdl
 - ドキュメントワークフロー起票用 Web サービスの WSDL ファイルです。
 BPM プロセスを定義する際に使用します。

ワークフロー連携サンプルプロジェクトをインポートすると予めプロジェクト内に含まれていますが、エンドポイントが「location="http://%IWP_HTTP_ADDRESS%:%IWP_HTTP_PORT%/imart/・・・"」となっている為、そのままでは使用出来ません。その為、インストールした iWP/iAF の環境にあわせてエンドポイントを修正してください。

♦ imart.xpath

● 認証ダイジェスト生成用カスタム xpath ファンクションです。 詳しくは、「BPM|Designer 操作ガイド 2.3.6 イントラマートへのログイン」を参照ください。

• reference.xform

処理結果参照用の X フォームです。 ドキュメントワークフローの処理結果を参照する際に使用します。 詳しくは、「BPM|Designer 操作ガイド 1.4.3 XForm を使用したヒューマンタスクの作成手順」を 参照ください。

♦ ResultInfo.xsd

処理結果返却用モデル定義 XML スキーマです。 ドキュメントワークフローから im-BPM へ処理結果を返却する際に、送信するメッセージに含まれる 各要素を定義しています。

処理結果仮判	デル定義(ResultI	nfoTyne)/†[/] T	「の要素が含まれます。
	/ / V AL #\$(INCOULU	moretime 1	マノ女 ポル・ロ みれしみ りっ

及達相不及為モノル 在義(Kesullillo Type) は以下の安宗が自よれる。		
correlationSetKey	相関セットキー(ドキュメントワークフロー)	
resultStatus	処理結果種別(ドキュメントワークフローの処理結果種別)	
resultMessage	処理結果メッセージ(ドキュメントワークフローの処理結果メッセージ)	
itemName	サンプルアプリケーション情報	
itemAmount	(サンプルで使用している「物品購買」の情報です。 im-BPM のプロセ	
itemPrice	ス開始時にアプリケーションデータを入力しているので、BPM Server	
itemTotal	内に同じデータを保持していますが、このサンプルではドキュメントワ	
deliveredPlace	ークフローの再申請時に、各項目の値が変更される可能性を考慮	
purchasePurpose	し、処理結果と共にアプリケーションデータも返却するようにしていま	
remarks	す。)	

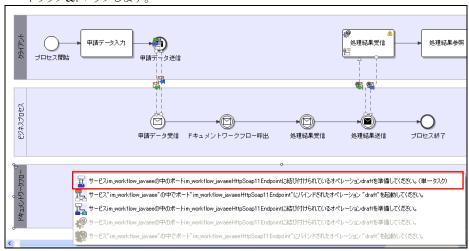
4.2.3.2 ワークフロー連携の設定

ここでは、ダイアグラム上でのワークフロー連携の設定箇所のみを説明します。

Xフォームの設定や詳しい BPM|Designer の操作方法については、「BPM|Designer 操作ガイド」を参照ください。

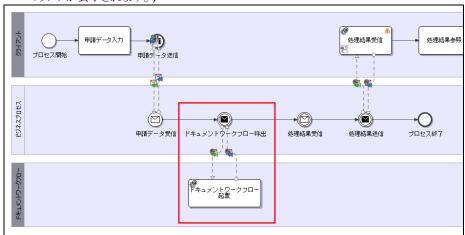
ワークフロー連携の設定は以下の手順で行っていきます。

① im_workflow_javaee.wsdl より draft オペレーションをプールに配置する。 プロセス・エクスプローラビューより「im_workflow_javaee.wsdl」→「im_workflow_javaee」→ 「im_workflow_javaeeHttpSoap11Endpoint」→「draft」を「ドキュメントワークフロー」プールへ ドラッグ&ドロップします。

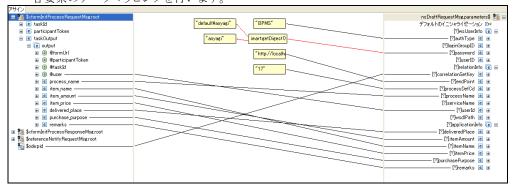


赤枠で囲われている「オペレーション draft を準備してください。(単一タスク)」を選択します。 「ドキュメントワークフロー」プールに「draft」という名前のタスクが配置されますので、その 「draft」タスクを選択し、プロパティビューよりラベルに「ドキュメントワークフロー起票」と入力し、 テクニカル・ネームに「draft」と入力します。

② 「ドキュメントワークフロー呼出」イベントと「ドキュメントワークフロー起票」タスクを紐付ける。 「ドキュメントワークフロー起票」タスクの配置が完了したら、「ドキュメントワークフロー呼出」イベント より、「ドキュメントワークフロー起票」タスクへ線を引きます。次に、「ドキュメントワークフロー起票」 タスクから「ドキュメントワークフロー呼出」イベントへ線を引きます。(線を引くと線上に自動でメッセージ のリンクが表示されます。)



③ マッパービューより各要素のデータマッピングを行う。 タスクの紐付けが完了したら、「ドキュメントワークフロー呼出」イベントを選択し、マッパービューから 各要素のデータマッピングを行います。



- wsUserInfo 認証用ユーザ情報
 - ◆ authType には認証タイプ「BPMS」を設定します。
 - ◆ password にはカスタム xpath ファンクションを使用し、認証ダイジェストを設定します。 詳しくは、「BPM|Designer 操作ガイド 2.3.6 イントラマートへのログイン」を参照してください。
- relationInfo ドキュメントワークフロー連携情報
 - ◆ correlationSetKey には「プロセスインスタンス ID」を設定します。 「プロセスインスタンス ID」の設定方法は「BPM|Designer 操作ガイド 2.5.4.5 相関セットキー値の マッピング」を参照ください。
 - ◆ endPoint にはドキュメントワークフロー処理結果を返却する際に呼び出すサービス(BPM|Server のサービス)のエンドポイントを指定します。エンドポイントは以下の規則で設定されます。
 - エンドポイント

http://%BPM|Server アドレス%:%BPM|Server ポート%/ode/process/%バンドルネーム%/ %ダイアグラム名%/%呼出先プールのテクニカルネーム%/%呼出元プールのテクニカルネーム%

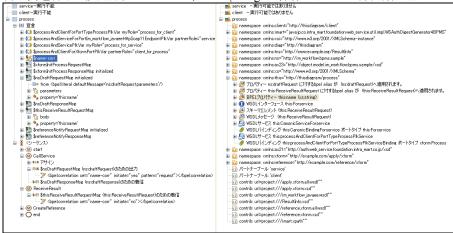
- ◆ processDefCd には起票を行うドキュメントワークフローのプロセス定義コードを設定します。 サンプルでは、サンプル「[カスタム]JavaEE 開発モデル・ドキュメントワークフロー1」の プロセス定義コード"17"を設定しています。
- ◆ processName にはドキュメントワークフローを起票する際に指定する「案件名」を指定します。 サンプルでは、Xフォームより入力されたプロセス名を設定しています。
- ◆ userId にはドキュメントワークフローの起票を行うユーザのユーザ ID を指定します。 サンプルでは、「\$xformInitProcessRequestMsg.root」→「taskOutput」→「output」→「@user」を 設定しています。これは、Xフォームからデータを入力し、プロセスを開始したユーザとなります。 また、「@user」には「%ログイングループ ID%¥%ユーザ ID%」の形式でユーザ情報を保持して いますので、iWP/iAFのサービス内でそのままユーザ ID として使用することは出来ません。
- applicationInfo
 - ◆ アプリケーションデータをマッピングします。サンプルでは、Xフォームより入力された申請データを各要素に設定しています。
- ④ 「完了」タスクと「処理結果受信」イベントを紐付け、スキーマ「ResultInfoType」を配置する。 次に、「ドキュメントワークフロー起票」タスクの次タスクとして、「完了」タスクを作成し、「完了」タスク から「処理結果受信」イベントへ線を引きます。線を引いたら、プロセス・エクスプローラビューより、 「ResultInfo.xsd」→「tns:ResultInfoType」を線上にドラッグ&ドロップします。



赤枠で囲われている「コンテンツとしてセットする。」を選択し、スキーマ・ResultInfoType を配置します。

⑤ 「correlationSetKey」をキーとして明示関連付けを行う。 最後に、データエディタービューより相関セットキーを設定し、明示関連付けを行います。

明示関連付けの設定方法については、「BPM|Designer 操作ガイド 2.5 非同期処理プロセス」を参照ください。



サンプルでは、SampleBpmnRelationInfoModel の correlationSetKey を相関セットキーとして設定を行っています。BPEL プロパティーの変数名は「name」、CorrelationSet の変数名は「name-corr」としています。

4.2.4 後処理プログラムの作成

ドキュメントワークフローの処理完了時に im-BPM へ処理結果を通知し、im-BPM のプロセスを再開する為、サンプルでは後処理プログラムを使用し、im-BPM へ処理結果を通知しています。

後処理プログラムの作成手順は以下の通りになります。

- ① BPM|Designer にて自動生成された WSDL を元にスタブクラスを作成する。
- ② スタブクラスを利用した Web サービス・クライアントを作成する。
- ③ 後処理プログラムから②で作成した Web サービス・クライアントを呼び出す。

4.2.4.1 スタブクラスの作成

スタブクラスの作成には Axis2 ツール「wsdl2java」コマンドを使用して作成したスタブを利用します。このコマンドは、Axis2 のバイナリーディストリビューションに付属しています。「java2WSDL」コマンドの詳細は、Axis2 プロジェクトの「Axis2 Reference Guide (http://ws.apache.org/axis2/1_4/reference.html)」ページのJava2WSDL Referenceを参照してください。

BPM|Designer を使用してダイアグラムを作成すると同一プロジェクト内の「build」ディレクトリに作成したダイアグラムの WSDL ファイルが自動生成されます。「wsdl2java」コマンドを使用し、この作成された WSDL ファイルよりスタブクラスを生成します。

■wsdl2java コマンド実行例

- > set AXIS2_HOME=C:\u00e4axis2-1.4
- > set BPM_WORKSPACE=C:\(\)BPM\(\)Designer\(\)\(\)sample_im_workflow_javaee\(\)\(\)build
- > set OUTPUT_DIR=C:\(\text{BPM\(\text{Y}\)Stub\)

wsdl2java -uri %BPM_WORKSPACE% ¥sample_im_workflow_javaee-BPM-Server.wsdl (← 実際は一行です) -o %OUTPUT DIR%

- -p sample.bpms.im_workflow.adb
- -sn CanonicServiceForservice

「%OUTPUT_DIR%¥sample.bpms.im_workflow.adb」ディレクトリに「CanonicServiceForserviceStub.java」と「CanonicServiceForserviceCallbackHandler.java」が生成されます。

「-sn CanonicServiceForservice」オプションを指定し、「sample_im_workflow_javaee-BPM-Server.wsdl」に定義されている「CanonicServiceForservice」サービスのスタブクラスのみを生成しています。これは BPM|Designer 上で「ビジネスプロセス」プールが、「ドキュメントワークフロー」プールと「クライアント」プール の2つのプールと紐付いており、「sample_im_workflow_javaee-BPM-Server.wsdl」には複数のサービスが 定義されているので、処理結果受信イベントを呼び出す為に必要なスタブクラスのみを生成する為です。 (「-sn CanonicServiceForservice」オプションをつけない場合、「CanonicServiceForservice」サービス以外の サービスのスタブクラスも同時に生成されます。)

サービス名は以下の規則で設定されます。

■サービス名

CanonicServiceFor%呼出元プールのテクニカルネーム% (X フォームの Web サービスの場合は、別の規則に基づいて設定されます。)

4.2.4.2 Webサービス・クライアントの生成

スタブクラスを生成したら、次にそのスタブクラスを利用した Web サービス・クライアントを生成します。 サンプルでは以下のクラスが Web サービス・クライアントの実装クラスとなります。

■Web サービス・クライアント実装クラスパス

%ApplicationRuntime%/doc/imart/WEB-INF/classes/sample/bpms/im_workflow/SampleBpwDraftServiceCallBack

SampleBpwDraftServiceCallBack.javaのソースは以下の通りです。

```
package sample.bpms.im workflow;
import sample.bpms.im workflow.adb.CanonicServiceForserviceStub;
import sample.bpms.im workflow.event.SelectSampleBpmnRelationEvent;
import sample.bpms.im workflow.event.SelectSampleBpmnRelationEventResult;
import jp.co.intra_mart.framework.base.event.EventManager;
import jp.co.intra_mart.framework.base.message.MessageManager;
import jp.co.intra mart.framework.base.util.UserInfo;
import jp.co.intra mart.framework.system.exception.ApplicationException;
import jp.co.intra mart.framework.system.exception.SystemException;
import jp.co.intra_mart.sample.bpw.purchase.common.model.object.CommonPurchaseModelObject;
import sample.bpms.im workflow.model.object.SampleBpmnRelationInfoModel;
public class SampleBpwDraftServiceCallBack {
   /* 処理結果種別 : 正常終了 */
   public static String STATUS COMPLATE = "1";
   /* 処理結果種別 : 途中終了 */
   public static String STATUS COMPULSION = "2";
    * 処理結果種別 : 取り止め */
   public static String STATUS CANCEL = "3";
   /* 処理結果種別 : 否認 */
   public static String STATUS REJECTION = "4";
   -
/* ユーザ情報 */
   private UserInfo userInfo;
   private SampleBpwDraftServiceCallBack() {
      super();
      userInfo = null;
   public SampleBpwDraftServiceCallBack(UserInfo userInfo) {
      this();
      this.userInfo = userInfo;
   public void callBackService(CommonPurchaseModelObject model, String resultStatus)
                                        throws SystemException, ApplicationException {
      SampleBpmnRelationInfoModel infoModel = getRelationInfo(model.getParameterCd());
      if(infoModel != null) {
          trv {
             CanonicServiceForserviceStub stub =
                new CanonicServiceForserviceStub(infoModel.getEndPoint());
             // 処理結果モデル生成
             CanonicServiceForserviceStub.ResultInfoType resultInfo =
                new CanonicServiceForserviceStub.ResultInfoType();
             resultInfo.setCorrelationSetKey(infoModel.getCorrelationSetKey());
             resultInfo.setResultStatus(resultStatus);
             resultInfo.setResultMessage(getResultMessage(resultStatus));
             resultInfo.setItemName(model.getItemName());
             resultInfo.setItemAmount(model.getItemAmount());
             resultInfo.setItemPrice(model.getItemPrice());
             resultInfo.setItemTotal(model.getItemTotal());
             resultInfo.setDeliveredPlace(model.getDeliveredPlace());
             resultInfo.setPurchasePurpose(model.getPurchasePurpose());
             resultInfo.setRemarks(model.getRemarks());
             // リクエスト生成
             CanonicServiceForserviceStub.ReceiveResultRequest request =
                new CanonicServiceForserviceStub.ReceiveResultRequest();
             request.setReceiveResultRequest(resultInfo);
             // リクエストの送信
             stub.ReceiveResult(request);
          } catch (Throwable e) {
             throw new SystemException(e.getMessage(), e);
      }
```

```
private SampleBpmnRelationInfoModel getRelationInfo(String parameterCd)
                                     throws SystemException, ApplicationException {
   EventManager manager = EventManager.getEventManager();
   SelectSampleBpmnRelationEvent event = (SelectSampleBpmnRelationEvent) manager.createEvent(
          "sample.bpms.im workflow.conf.bpms im workflow", "select bpmn relation", userInfo);
   // eventに検索条件を設定
   event.setParameterCd(parameterCd);
   // event実行
   SelectSampleBpmnRelationEventResult eventResult =
      (SelectSampleBpmnRelationEventResult) manager.dispatch(event, true);
   return eventResult.getInfoModel();
private String getResultMessage(String resultStatus) throws SystemException {
   String message = "";
   MessageManager manager = MessageManager.getMessageManager();
   \textbf{if} (\texttt{resultStatus.equals} (\texttt{SampleBpwDraftServiceCallBack}. \textit{STATUS\_COMPLATE})) \  \  \{ \texttt{complete} \} \\
      // 正常終了時のメッセージを取得
      message = manager.getMessage("sample.bpms.im workflow.conf.BpmsImWorkflow",
                                      "bpmn.result complate", null);
   } else if(resultStatus.equals(SampleBpwDraftServiceCallBack.STATUS COMPULSION)) {
       // 途中終了時のメッセージを取得
      message = manager.getMessage("sample.bpms.im workflow.conf.BpmsImWorkflow",
                                      "bpmn.result compulsion", null);
   } else if(resultStatus.equals(SampleBpwDraftServiceCallBack.STATUS CANCEL)) {
       // 取り止め時のメッセージを取得
      message = manager.getMessage("sample.bpms.im workflow.conf.BpmsImWorkflow",
                                      "bpmn.result_cacel", null);
   } else if(resultStatus.equals(SampleBpwDraftServiceCallBack.STATUS REJECTION)) {
       // 否認時のメッセージを取得
      message = manager.getMessage("sample.bpms.im workflow.conf.BpmsImWorkflow",
                                      "bpmn.result rejection", null);
   return message;
```

■ callBackService メソッド

スタブクラスを使用し、処理結果を BPM|Server の「処理結果受信」イベントへ返却します。

① スタブクラスのインスタンス生成 まずは、スタブクラスのインスタンスを生成します。

CanonicServiceForserviceStub stub =

 $new\ Canonic Service For service Stub (info Model.get End Point ());$

ここで、引数にワークフロー連携情報エンドポイントを指定していますが、これはサンプル内に予め用意されているスタブクラスのエンドポイントは「http://localhost:8080/ode/・・・」となっている為、BPM|Designerで設定した BPM|Server のエンドポイントにメッセージを送信する為です。

② 処理結果の設定 次に、ドキュメントワークフローの処理結果返却用のモデルを生成します。

CanonicServiceForserviceStub.ResultInfoType resultInfo = new CanonicServiceForserviceStub.ResultInfoType();

BPM|Designer で「完了」タスクと「処理結果受信」イベントを紐付ける線上に設定したスキーマ「ResultInfoType」の XML スキーマに従ったモデルを生成し、各要素の値を設定していきます。

③ リクエストの送信 最後にリクエストを生成し、BPM|Server に送信します。

// リクエスト生成

CanonicServiceForserviceStub.ReceiveResultRequest request = new CanonicServiceForserviceStub.ReceiveResultRequest(): request.setReceiveResultRequest(resultInfo); // リクエストの送信

stub.ReceiveResult(request);

リクエストの送信時の「ReceiveResult」は「処理結果受信」イベントのテクニカルネームとなります。 また、リクエスト送信時の「ReceiveResultRequest」は「処理結果受信」イベントの着信メッセージ変数名 となります。(サンプルでは、「処理結果受信」イベントのテクニカルネームに「ReceiveResult」を指定し、 着信メッセージ変数名はデフォルトの設定のままとしてあります。)

getRelationInfo メソッド

ドキュメントワークフロー連携情報を取得する為のメソッドです。 イベント内で、ドキュメントワークフロー起票時に登録されたワークフロー連携情報を取得します。

イベント内の処理の実装例は以下のソースを参照してください。

起票イベント実装クラスパス

%ApplicationRuntime%/doc/imart/WEB-INF/classes/sample/bpms/im workflow/event/ Select Sample Bpmn Relation Event Listener. java

getResultMessage メソッド

処理結果メッセージを取得する為のメソッドです。 処理結果種別に応じたメッセージを取得する為に使用します。

メッセージは以下のファイルに配置されています。

ワークフロー連携サンプル(JavaEE 開発モデル)メッセージファイル %ApplicationRuntime%/doc/imart/WEB-INF/classes/sample/bpms/im workflow/conf/ MessageConfig BpmsImWorkflow ja.properties

後処理プログラムからWebサービス・クライアントの呼び出し 4.2.4.3

Web サービス・クライアントの生成したら、最後にドキュメントワークフローの後処理プログラム内から、生成した Web サービス・クライアントを呼び出します。

サンプルでは以下のクラスが後処理プログラムの実装クラスとなります。

後処理プログラム実装クラスパス

%ApplicationRuntime%/doc/imart/WEB-INF/classes/jp/co/intra mart/sample/bpw/purchase/appendix/ PostProcessing

%ApplicationRuntime%/doc/imart/WEB-INF/classes/jp/co/intra mart/sample/bpw/purchase/appendix/ ApproverPostProcessing

実装例は以下の通りです。

if(modelObject.getEndFlag() != null && modelObject.getEndFlag().equals("1")) { // SampleDraftService から申請処理をされた案件は CallBack プログラムを実行する。 SampleBpwDraftServiceCallBack callback = new SampleBpwDraftServiceCallBack(userInfo); Callback.callBackService(modelObject, %処理結果種別%);

ドキュメントワークフローの完了には以下のような処理が想定されるので、その処理毎の後処理プログラムで Web サービス・クライアントの呼び出しを行う必要があります。

処理	実装箇所	
完了(通常)	End 時実行プログラム内の proceed メソッド	
完了(途中終了)	途中終了可能なタスクの後処理プログラム内の compulsion メソッド	
取り止め	起票タスクの後処理プログラム内の compulsion メソッド	
否認	起票タスク以外の各タスクの後処理プログラム内の rejection メソッド	

4.3 スクリプト開発モデル

ここでは、スクリプト開発モデルを用いたワークフロー連携方法をサンプル(sample_im_workflow_script)を使用して説明します。sample_im_workflow_script については、「BPM イントロダクション 2.1.3 ワークフロー連携サンプルプロジェクト(スクリプト開発モデル)」を参照ください。

ここでは連携に必要な以下の手順について説明を行います。

- ◆ ドキュメントワークフローを起票する Web サービスを作成する。
- ◆ iWP/iAF の ApplicationRuntime へ Web サービスをデプロイする。
- ◆ BPM|Designer を使用し、ビジネス・プロセス・ダイアグラムを作成する。
- ◆ IWP/iAF の StorageService 〜ダイアグラム作成時に自動生成される WSDL ファイルを配置する。
- ◆ ドキュメントワークフローの処理結果を返却する後処理プログラムを作成する。

4.3.1 Webサービスの作成

まずは、ドキュメントワークフローを起票する為の Web サービスを作成します。 サンプルでは以下のクラスが Web サービスの実装クラスとなります。

■Web サービスとして公開する JavaScript クラスパス

%ResourceService%/pages/src/sample/bpms/im workflow/bpw draft service.js

■JavaScript ラッパークラスパス

 $\% \ Application Runtime \%/doc/imart/WEB-INF/classes/sample/bpms/im_workflow/Sample BpwDraftService For Java Script.java$

ここでは、Web サービスとして公開する JavaScript クラスの説明のみを行います。 スクリプト開発モデルでの Web サービスの作成方法や JavaScript ラッパークラスについては、「Web サービス・プログラミングガイド 4.1 Web サービス・プロバイダの作成」を参照ください。

bpw_draft_service.js のソースは以下の通りです。

```
var drafter = new WkfDrafter(user_info[1], user_info[0]);
var processDefList =
    drafter.getActivityProcessDefList(serch_condition, null, 0, 0);
if( processDefList == null || processDefList.length == 0 ) {
    // 起票可能一覧が取得できないので終了する。
    logger.error("The specified process definition doesn't exist.");
var processDef = processDefList[0];
// ワークフローユーザトランザクションの準備
var ut = new UserTransaction();
if( ut == null ) {
    // UserTransactionオブジェクト取得失敗
    return "";
.
// ワークフローユーザトランザクションの開始
if( !ut.begin() ) {
    // トランザクションの開始に失敗
    return "";
// アプリケーション情報の登録
var result = insertApplicationInfo(parameter_cd, applicationInfo);
if( result.error ) {
    // アプリケーション情報登録失敗
    ut.rollback();
    logger.error(result.errorMessage);
    return "";
}
// BPMS連携情報の登録
result = insertRelationInfo(parameter_cd, relationInfo);
if( result.error ) {
    // BPMS連携情報登録失敗
    ut.rollback();
    logger.error(result.errorMessage);
    return "";
}
// 所属組織情報の取得
var\ departmentList = drafter.getTargetDepartment
    (processDef.process_def_cd, processDef.version_cd, processDef.activity_cd, "");
if(departmentList == null || departmentList.length == 0) {
    // 所属組織情報が取得できないので終了する。
    ut.rollback();
    logger.error("Belonging organization information cannot be acquired.");
    return "":
var departmentInfo = departmentList[0];
// ワークフロー申請処理実行
var app_key = new Array();
app_key[0] = parameter_cd;
drafter.noTransaction();
result = drafter.draft(processDef.process_def_cd,
                         processDef.version_cd,
                         relationInfo.processName,
                         app_key,
                         departmentInfo.company cd,
                         departmentInfo.department\_cd,
                         null,
                         null,
                         null);
if( !result.isSuccess ) {
    // 申請処理失敗時
    ut.rollback();
```

```
logger.error(result.getMessage);
        return "";
    // コミット
    if(!ut.commit()) {
        // コミット失敗
        return "";
    return result.processInfo.process cd;
 * アプリケーションデータをDBへ登録します。
function insertApplicationInfo(parameter cd, applicationInfo) {
    // データ設定
    var oItem
                              = new Object();
                             = parameter_cd;
    oItem.parameter_cd
                              = applicationInfo.itemName;
    oItem.item_name
    oItem.item\_amount
                              = applicationInfo.itemAmount;
                            = applicationInfo.itemPrice;
    oItem.item price
                            = ( Number( oItem.item_amount ) * Number( oItem.item_price ) ) + "";
    oItem.item_total
    oItem.delivered_place = applicationInfo.deliveredPlace;
    oItem.purchase_purpose = applicationInfo.purchasePurpose;
    oItem.remarks
                             = applicationInfo.remarks;
                            = "0";
= "1";
    oItem.item_flag
    oItem.end_flag
    // insert実行
    return DatabaseManager.insert( "sample_b_bpw_t_purchase", oItem);
*BPMS連携情報をDBへ登録します。
function insertRelationInfo(parameter_cd, relationInfo) {
    // データ設定
    var oItem
                                = new Object();
    oItem.parameter_cd
                               = parameter_cd
    oItem.correlation_set_key = relationInfo.correlationSetKey;
                               = relationInfo.endPoint;
    oItem.end_point
    oItem.wsdl_path
                               = relationInfo.wsdlPath;
    oItem.service name
                               = relationInfo.serviceName;
    // insert実行
    return DatabaseManager.insert( "sample b bpw t bpmn relation", oItem);
```

4.3.1.1 引数に指定されている各要素の定義

この Web サービスで使用する要素は以下の通りです。

① relationInfo サンプル連係情報 im-BPM との連携情報を保持します。この要素内の各子要素にワークフロー連携を行う為に 必要な値を保持します。

relationInfo の子要素は以下の通りになります。

relation mio v 1 文宗は外 「v 起力になりな)。		
correlationSetKey	相関セットキー	im-BPM との連携用のキー
endpoint	エンドポイント	BPM Server のエンドポイント
wsdlPath	WSDL ファイル配置パス	im-BPM の WSDL 格納先のパス
serviceName	サービス名	処理結果を返却する時に呼び出すサービスの名前
processDefCd	プロセス定義コード	起票するドキュメントワークフローのプロセス定義コード
processName	プロセス名	起票するドキュメントワークフローの案件名
userId	ユーザ ID	起票を行うユーザのユーザ ID

相関セットキーは、im-BPM とドキュメントワークフローの関連付けを行う為のキーになります。サンプルでは、im-BPM のプロセスインスタンス ID をキーとしています。

エンドポイントには、ドキュメントワークフロー完了時に処理結果を BPM|Server へ返却する際に呼び出される BPM|Server のサービスのエンドポイントになります。

WSDL ファイル配置パス・サービス名は Web サービスクライアント生成時にアプリケーション共通モジュール API SOAPClient オブジェクトを使用する際に利用します。 SOAPClient オブジェクトについては、「Web サービス・プログラミングガイド 4.2 Web サービス・クライアントの作成」および「API リスト」を参照ください。

② applicationInfo サンプルアプリケーション情報

サンプルアプリケーションのアプリケーションデータを保持します。

この例では、ドキュメントワークフローのサンプル

[カスタム]JavaEE 開発モデル・ドキュメントワークフロー1]

と連携しているので、このサンプルアプリケーションの起票に必要な情報を保持しています。 このサンプルは、「物品購買」のサンプルアプリケーションとなっていますので、物品購買の為の 要素が定義されています。

applicationInfoの子要素は以下の通りになります。

11		
itemName	品名	品名
itemAmount	数量	購入数
itemPrice	金額	購入品の単価
deliveredPlace	納品場所	納品場所
purchasePurpose	購入目的	購入目的
remarks	備考	備考

4.3.1.2 起票ユーザ情報の設定

このサンプルでは、プロセスを開始したユーザを起票ユーザとする為、プロセスの開始ユーザ情報が relationInfo の userId 要素に受け渡されます。

しかし、im-BPM のユーザ情報は「%ログイングループ ID%¥%ユーザ ID%」の形式になっている為、そのままでは使用出来ません。

その為、ログイングループ ID とユーザ ID を切り離す必要があります。

実装例

// 起票ユーザ情報取得

var user_info = relationInfo.userId.split("\footnote{x\footnote{y\finter\finter{y\finter\finty}}}}}}}}}}}}}}}}}}}}

4.3.1.3 ドキュメントワークフローの起票処理

ドキュメントワークフローの起票には、ワークフローモジュール API を使用します。

ワークフローモジュール API については、「API リスト」および「ワークフロー プログラミングガイド 3.3 API を使用する」を参照ください。

ドキュメントワークフローの起票には以下のような処理を行っています。

- ◆ 指定されたプロセス定義が申請可能かチェックする。
- ◆ アプリケーション情報を DB へ登録する。
- ◆ im-BPM 連係情報を DB へ登録する。
- ◆ 起票時の所属組織情報を取得する。
- ◆ ドキュメントワークフローの起票 API を実行する。

4.3.2 Webサービスのデプロイ

ドキュメントワークフロー起票用の Web サービスの作成が完了したら、

iWP/iAF の ApplicationRuntime 上へ Web サービスをデプロイします。

サンプルでは、「services.xml」を規定のディレクトリに配置する方法でデプロイを行っています。

Web サービスのデプロイ方法については「Web サービス・プログラミングガイド 6.4 Web サービスのデプロイ方法」を参照ください。

■ services.xml の配置ディレクトリ

%ApplicationRuntime%/doc/imart/WEB-INF/services/im_bpms_samples

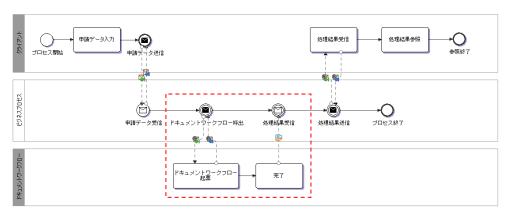
「services.xml」の内容は以下の通りです。

```
19: <service name="im_workflow_script" >
     <Description></Description>
21:
22:
      <parameter name="ServiceClass">sample.bpms.im workflow.SampleBpwDraftServiceForJavaScript/parameter>
23:
24:
     <module ref="im_ws_auth"/>
25:
26:
     <messageReceivers>
27:
        <messageReceiver
                  mep="http://www.w3.org/2004/08/wsdl/in-only"
                  class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
28:
                  mep="http://www.w3.org/2004/08/wsdl/in-out"
                  class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"/>
     </messageReceivers>
30: </service>
```

4.3.3 ビジネス・プロセス・ダイアグラムの作成

サンプルでは、以下のダイアグラムを作成しています。

ワークフロー連携サンプルプロジェクト(スクリプト開発モデル)については「BPM イントロダクション 2.1.3 ワークフロー連携サンプルプロジェクト(スクリプト開発モデル)」を参照ください。



点線で囲まれた範囲がワークフローとの連携の部分になります。

4.3.3.1 ワークフロー連携サンプル(スクリプト開発モデル)に含まれるファイル

sample im workflow script には以下のファイルが含まれます。

- sample im workflow script.bpm
 - ◆ ワークフロー連携サンプルのビジネス・プロセス・ダイアグラムです。BPM プロセスの定義等が設定されています。

■ apply.xform

◆ 申請用の X フォームです。

申請データを入力し、プロセスを開始する為に使用します。

詳しくは、「BPM|Designer 操作ガイド 1.4.3 XForm を使用したヒューマンタスクの作成手順」を参照ください。

im_workflow_script.wsdl

◆ ドキュメントワークフロー起票用 Web サービスの WSDL ファイルです。

BPM プロセスを定義する際に使用します。

ワークフロー連携サンプルプロジェクトをインポートすると予めプロジェクト内に含まれていますが、エンドポイントが「location="http://%IWP_HTTP_ADDRESS%:%IWP_HTTP_PORT%/imart/・・・"」となっている為、そのままでは使用出来ません。その為、インストールした iWP/iAF の環境にあわせてエンドポイントを修正してください。

■ imart.xpath

◆ 認証ダイジェスト生成用カスタム xpath ファンクションです。 詳しくは、「BPM|Designer 操作ガイド 2.3.6 イントラマートへのログイン」を参照ください。

■ reference.xform

◆ 処理結果参照用の X フォームです。

ドキュメントワークフローの処理結果を参照する際に使用します。

詳しくは、「BPM|Designer 操作ガイド 1.4.3 XForm を使用したヒューマンタスクの作成手順」を参照ください。

■ ResultInfo.xsd

◆ 処理結果返却用モデル定義 XML スキーマです。

ドキュメントワークフローから im-BPM へ処理結果を返却する際に、送信するメッセージに含まれる 各要素を定義しています。

処理結果返却モデル定義(ResultInfoType)は以下の要素が含まれます。

工相水色料 c / /		
correlationSetKey	相関セットキー(ドキュメントワークフロー)	
resultStatus	処理結果種別(ドキュメントワークフローの処理結果種別)	
resultMessage	処理結果メッセージ(ドキュメントワークフローの処理結果メッ	
	セージ)	
itemName	サンプルアプリケーション情報	
itemAmount	(サンプルで使用している「物品購買」の情報です。im-BPM	
itemPrice	のプロセス開始時にアプリケーションデータを入力している	
itemTotal	ので、BPM Server内に同じデータを保持していますが、この	
deliveredPlace	サンプルではドキュメントワークフローの再申請時に、各項	
purchasePurpose	目の値が変更される可能性を考慮し、処理結果と共にアプ	
remarks	リケーションデータも返却するようにしています。)	

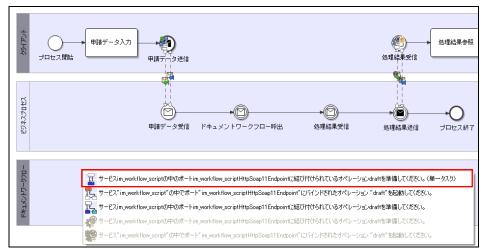
4.3.3.2 ワークフロー連携の設定

ここでは、ダイアグラム上でのワークフロー連携の設定箇所のみを説明します。

Xフォームの設定や詳しい BPM|Designer の操作方法については、「BPM|Designer 操作ガイド」を参照ください。

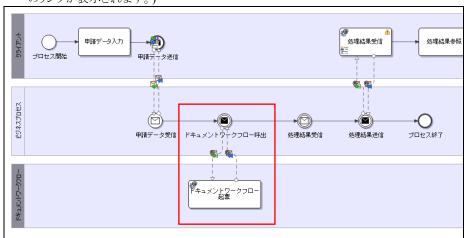
ワークフロー連携の設定は以下の手順で行っていきます。

① im_workflow_script.wsdl より draft オペレーションをプールに配置する。 プロセス・エクスプローラビューより「im_workflow_script.wsdl」→「im_workflow_script」→ 「im_workflow_scriptHttpSoap11Endpoint」→「draft」を「ドキュメントワークフロー」プールへ ドラッグ&ドロップします。

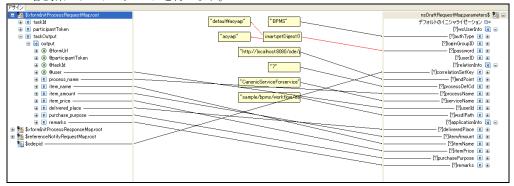


赤枠で囲われている「オペレーション draft を準備してください。(単一タスク)」を選択します。 「ドキュメントワークフロー」プールに「draft」という名前のタスクが配置されますので、その 「draft」タスクを選択し、プロパティビューよりラベルに「ドキュメントワークフロー起票」と入力し、 テクニカル・ネームに「draft」と入力します。

② 「ドキュメントワークフロー呼出」イベントと「ドキュメントワークフロー起票」タスクを紐付ける。 「ドキュメントワークフロー起票」タスクの配置が完了したら、「ドキュメントワークフロー呼出」イベント より、「ドキュメントワークフロー起票」タスクへ線を引きます。次に、「ドキュメントワークフロー起票」 タスクから「ドキュメントワークフロー呼出」イベントへ線を引きます。(線を引くと線上に自動でメッセージ のリンクが表示されます。)



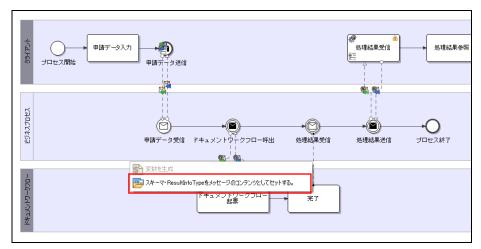
③ マッパービューより各要素のデータマッピングを行う。 タスクの紐付けが完了したら、「ドキュメントワークフロー呼出」イベントを選択し、マッパービューから 各要素のデータマッピングを行います。



- wsUserInfo 認証用ユーザ情報
 - ◆ authType には認証タイプ「BPMS」を設定します。
 - ◆ password にはカスタム xpath ファンクションを使用し、認証ダイジェストを設定します。 詳しくは、「BPM|Designer 操作ガイド 2.3.6 イントラマートへのログイン」を参照してください。
- relationInfo ドキュメントワークフロー連携情報
 - ◆ correlationSetKey には「プロセスインスタンス ID」を設定します。 「プロセスインスタンス ID」の設定方法は「BPM|Designer 操作ガイド 2.5.4.5 相関セットキー値の マッピング」を参照ください。
 - ◆ endPoint にはドキュメントワークフロー処理結果を返却する際に呼び出すサービス(BPM|Server のサービス)のエンドポイントを指定します。エンドポイントは以下の規則で設定されます。
 - エンドポイント
 http://%BPM|Server アドレス%:%BPM|Server ポート%/ode/process/%バンドルネーム%/
 %ダイアグラム名%/%呼出先プールのテクニカルネーム%/%呼出元プールのテクニカルネーム%
 - ◆ processDefCd には起票を行うドキュメントワークフローのプロセス定義コードを設定します。 サンプルでは、サンプル「[カスタム]スクリプト開発モデル・ドキュメントワークフロー1」の プロセス定義コード"7"を設定しています。
 - ◆ processName にはドキュメントワークフローを起票する際に指定する「案件名」を指定します。 サンプルでは、Xフォームより入力されたプロセス名を設定しています。
 - ◆ userId にはドキュメントワークフローの起票を行うユーザのユーザ ID を指定します。 サンプルでは、「\$xformInitProcessRequestMsg.root」→「taskOutput」→「output」→「@user」を 設定しています。これは、X フォームからデータを入力し、プロセスを開始したユーザとなります。 また、「@user」には「%ログイングループ ID%¥%ユーザ ID%」の形式でユーザ情報を保持して いますので、iWP/iAF のサービス内でそのままユーザ ID として使用することは出来ません。
 - ◆ serviceName にはドキュメントワークフローの処理結果を返却する際に呼び出すサービスのサービス名を指定します。サービス名は以下の規則で設定されます。
 - サービス名
 CanonicServiceFor%呼出元プールのテクニカルネーム%
 - ◆ wsdlPath には BPM|Designer で自動生成された WSDL ファイルを iWP/iAF に配置した際の WSDL ファイルを指定します。WSDL ファイルの配置については、「5.3.4 WSDL ファイルの配置」 で説明します。

サンプルでは、「sample/bpms/workflow/sample_im_workflow_script-process.wsdl」を設定しています。

- applicationInfo
 - アプリケーションデータをマッピングします。サンプルでは、Xフォームより入力された申請データを各要素に設定しています。
- ④ 「完了」タスクと「処理結果受信」イベントを紐付け、スキーマ「ResultInfoType」を配置する。 次に、「ドキュメントワークフロー起票」タスクの次タスクとして、「完了」タスクを作成し、「完了」タスク から「処理結果受信」イベントへ線を引きます。線を引いたら、プロセス・エクスプローラビューより、 「ResultInfo.xsd」→「tns:ResultInfoType」を線上にドラッグ&ドロップします。



赤枠で囲われている「コンテンツとしてセットする。」を選択し、スキーマ・ResultInfoType を配置します。

⑤ 「correlationSetKey」をキーとして明示関連付けを行う。 最後に、データエディタービューより相関セットキーを設定し、明示関連付けを行います。

明示関連付けの設定方法については、「BPM|Designer 操作ガイド 2.5 非同期処理プロセス」を参照ください。



サンプルでは、relationInfoの correlationSetKey を相関セットキーとして設定を行っています。 BPEL プロパティーの変数名は「name」、CorrelationSet の変数名は「name-corr」としています。

WSDLファイルの配置 4.3.4

ビジネス・プロセス・ダイアグラムの作成が完了したら、BPM|Designer で自動生成された WSDL ファイルを iWP/iAFの Storage Service 上に配置します。これは、ドキュメントワークフローの処理結果を返却する為の スタブクラスをアプリケーション共通マスタ API SOAPClient オブジェクトを使用して自動生成するためです。

SOAPClient オブジェクトでは、wsdlUrl から WSDL を指定する方法と、Storage Service 上に存在する WSDL ファイルを利用する方法がありますが、BPM|Server で使用している Axis2 のバージョンが 1.3 になっている為、 wsdlUrl から WSDL を指定する方法では schemaLocation が解決できず、スタブクラスの生成に失敗してしまい ます。

その為、サンプルでは Storage Service 上に存在する WSDL ファイルを利用する方法を使用しますので、 WSDL ファイルを iWP/iAF の Storage Service へ配置します。

また、WSDL ファイルを配置する際に、その WSDL ファイルを解析する為に必要となる WSDL ファイルおよび XML スキーマファイルも同じディレクトリ内へ配置する必要があります。もし、定義されているファイルが配置さ れていない場合、SOAPClientオブジェクトのインスタンス生成時にエラーとなりますので、ご注意ください。

iWP/iAFへ配置するWSDLファイル名は以下ような名前のファイルとなります。

■ WSDL ファイルのファイル名

%BPM|Designer ワークスペース%/プロジェクト名/build/

%ダイアグラム名%-%呼出先プールのテクニカルネーム%.wsdl

サンプルでは、以下の WSDL ファイルを StorageService 上へ配置します。

- ワークフロー連携サンプル(スクリプト開発モデル)の WSDL ファイル名 %BPM|Designer ワークスペース%/sample_im_workflow_script/build/sample_im_workflow_script-process.wsdl
- sample_im_workflow_script-process.wsdl 解析時に必要となるファイル
 - apply.xform.all.wsdl
 - apply.xform.xsd
 - im workflow script.wsdl
 - ◆ reference.xform.all.wsdl
 - reference.xform.xsd
 - ♦ ResultInfo.xsd
- WSDL ファイルの配置先パス
 %Storage Service%/storage/sample/bpms/workflow

WSDL ファイル配置先パスは BPM|Designer でデータマッピングを行った際に指定した wsdlPath となります。

4.3.5 後処理プログラムの作成

ドキュメントワークフローの処理完了時に im-BPM へ処理結果を通知し、im-BPM のプロセスを再開する為、サンプルでは後処理プログラムを使用し、im-BPM へ処理結果を通知しています。

後処理プログラムの作成手順は以下の通りになります。

- ① Web サービス・クライアントを作成する。
- ② 後処理プログラムから②で作成した Web サービス・クライアントを呼び出す。

4.3.5.1 Webサービス・クライアントの生成

まず、SOAPClient オブジェクトを使用して Web サービス・クライアントを生成します。 サンプルでは以下のクラスが Web サービス・クライアントの実装クラスとなります。

■ Web サービス・クライアント実装クラスパス %Resource Service%/pages/src/sample/bpms/im_workflow/callback.js callback.js のソースは以下の通りです。

```
function callbackService(data, result_status) {
    // BPM連携情報取得
    var relationInfo = getRelationInfo(data.parameter_cd);
    if( relationInfo.error ) {
        var logger = Logger.getLogger();
        logger.error(relationInfo.errorMessage);
        return;
    }
    if( relationInfo.countRow == 0 ) {
        return;
    }
    relationInfo = relationInfo.data[0];

    var wsdl = new VirtualFile(relationInfo.wsdl_path);
    var soapClient = new SOAPClient(wsdl, relationInfo.service_name, null, relationInfo.end_point);
```

```
// ステップ2:Webサービスを呼び出すソースコードのサンプルを表示
    //**********************
    //soapClient.getSampleCode();
    // Sample Data : 'receiveResultRequest'
    var receiveResultRequest =
    /* Object <ResultInfoType> */
        "itemTotal" : data.item_total,
"itemPrice" : data.item_price,
        "itemAmount" : data.item_amount,
         "remarks" : data.remarks,
        "itemName" : data.item_name,
        "correlationSetKey": relationInfo.correlation_set_key,
        "resultMessage" : getResultMessage(result_status),
"deliveredPlace" : data.delivered_place,
        "purchasePurpose" : data.purchase_purpose,
         "resultStatus" : result_status
    };
    // Webサービスの呼び出し
    soapClient.ReceiveResult(receiveResultRequest);
function getRelationInfo(parameter_cd) {
    var sql = "SELECT * '
             + "FROM sample_b_bpw_t_bpmn_relation"
+ "WHERE parameter_cd = "" + parameter_cd + """;
    return DatabaseManager.select(sql);
function getResultMessage(result_status) {
    var message = "";
    if(result_status == "1") {
        // 正常終了時のメッセージを取得
        message = MessageManager.getMessage("BPM.workflow.result\_complate");
    } else if(result_status == "2") {
        // 途中終了時のメッセージを取得
        message = MessageManager.getMessage("BPM.workflow.result_compulsion");
    } else if(result status == "3") {
        // 取り止め時のメッセージを取得
        message = MessageManager.getMessage("BPM.workflow.result\_cacel");
    } else if(result_status == "4") {
        // 否認時のメッセージを取得
        message = MessageManager.getMessage("BPM.workflow.result_rejection");
    return message;
```

■ callbackService メソッド

SOAPClient オブジェクトを使用して、処理結果を BPM|Server の「処理結果受信」イベントへ返却します。 SOAPClient オブジェクトについては、「Web サービス・プログラミングガイド 4.2 Web サービス・クライアントの作成」および「API リスト」を参照ください。

① SOAPClient オブジェクトのインスタンス生成 まずは、SOAPClient オブジェクトのインスタンスを生成します。

var wsdl = new VirtualFile(relationInfo.wsdl_path);
var soapClient = new SOAPClient(wsdl, relationInfo.service_name, null, relationInfo.end_point);

ここで、引数にワークフロー連携情報のサービス名およびエンドポイントを指定していますが、サービス名は WSDL ファイルに複数のサービスが設定されている場合にサービス名を指定しないと、SOAPClient オブジクトが正常にスタブクラスを生成できないためです。またエンドポイントを指定しているのは、BPM|Designer で設定した BPM|Server のエンドポイントにメッセージを送信する為です。

② 処理結果の設定

次に、ドキュメントワークフローの処理結果返却用のモデルを生成します。 Web サービスを呼び出すソースコードのサンプルを表示するには、SOAPClient.getSampleCode()を使用します。

BPM|Designer で「完了」タスクと「処理結果受信」イベントを紐付ける線上に設定したスキーマ「ResultInfoType」の XML スキーマに従ったオブジェクトを生成し、各要素の値を設定していきます。 変数名の「ReceiveResultRequest」は「処理結果受信」イベントの着信メッセージ変数名となります。

③ Web サービスの呼び出しBPM|Server の Web サービスを呼び出します。

```
//------
// Web サービスの呼び出し
//-----soapClient.ReceiveResult(receiveResultRequest);
```

Web サービス呼び出し時の「ReceiveResult」は「処理結果受信」イベントのテクニカルネームとなります。 (サンプルでは、「処理結果受信」イベントのテクニカルネームに「ReceiveResult」を指定し、 着信メッセージ変数名はデフォルトの設定のままとしてあります。)

■ getRelationInfo メソッド

ドキュメントワークフロー連携情報を取得する為のメソッドです。
ドキュメントワークフロー起票時に登録されたワークフロー連携情報を取得します。

■ getResultMessage メソッド

処理結果メッセージを取得する為のメソッドです。 処理結果種別に応じたメッセージを取得する為に使用します。

メッセージは以下のファイルに配置されています。

▽ ワークフロー連携サンプル(スクリプト開発モデル)メッセージファイル %Server Manager%/conf/message/bpmsample-message_ja.properties

4.3.5.2 後処理プログラムからWebサービス・クライアントの呼び出し

Web サービス・クライアントの生成したら、最後にドキュメントワークフローの後処理プログラム内から、生成した Web サービス・クライアントを呼び出します。

サンプルでは以下のクラスが後処理プログラムの実装クラスとなります。

■ 後処理プログラム実装クラスパス

%Resource Service%/pages/src/sample/bpw/purchase/appendix/post_processing.js %Resource Service%/pages/src/sample/bpw/purchase/appendix/approver_post_processing.js

実装例は以下の通りです。

```
if(result.data[0].end_flag == "1") {
//BPM 連携プロセスの場合は CallBack プログラムを実行する。
Content.executeFunction("sample/bpms/im_workflow/callback", "callbackService", result.data[0], "%処理種別%");
}
```

ドキュメントワークフローの完了には以下のような処理が想定されるので、その処理毎の後処理プログラムで Web サービス・クライアントの呼び出しを行う必要があります。

処理	実装箇所
完了(通常)	End 時実行プログラム内の proceed メソッド
完了(途中終了)	途中終了可能なタスクの後処理プログラム内の compulsion メソッド
取り止め	起票タスクの後処理プログラム内の compulsion メソッド
否認	起票タスク以外の各タスクの後処理プログラム内の rejection メソッド

5 API 使用例

5.1 はじめに

ここでは、IM-BPMのAPI使用例について説明します。以下のJavaEE開発モデルで利用するクラスと、スクリプト開発モデルで利用するオブジェクトの使用例について説明します。

■ JavaEE 開発モデル

NoticeProgressManager	進捗通知マネージャクラス
TaskManager	タスクマネージャクラス
TokenManager	トークンマネージャクラス

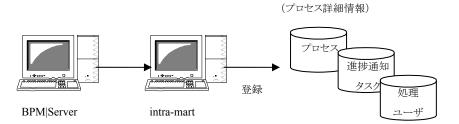
■ スクリプト開発モデル

NoticeProgressManager	進捗通知マネージャオブジェクト
TaskManager	タスクマネージャオブジェクト
TokenManager	トークンマネージャオブジェクト

5.2 NoticeProgressManager

5.2.1 プロセス詳細情報の登録

プロセス詳細情報とは、ユーザが業務プロセスの進捗状況を確認ための情報です。ここではプロセス詳細情報の 登録を行います。



■ JavaEE 開発モデル

```
// プロセスインスタンス ID
String processInstanceId = "169875";
             -- 処理ユーザ情報を設定する --
// 処理ユーザのリスト
List<BpmsUser> userList = new ArrayList<BpmsUser>();
// 処理ユーザのインスタンス生成
BpmsUser user = new BpmsUser();
// ユーザCD
user.setUserCd("aoyagi");
// 処理ユーザをリストにセット
userList.add(user);
     ----- 進捗通知タスク情報を設定する --
// 進捗通知タスクのリスト
List<NoticeProgressTask> noticeTaskList = new ArrayList<NoticeProgressTask>();
// 進捗通知タスクのインスタンス生成
NoticeProgressTask noticeTask = new NoticeProgressTask();
// 進捗通知タスクID
noticeTask.setNoticeTaskId("task1");
// 進捗通知タスク名
noticeTask.setNoticeTaskName("タスク名");
// プロセスインスタンスID
noticeTask.setProcessInstanceId(processInstanceId);
// 進捗通知タスクステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」:BPMS_STATUS_COMPLETE
notice Task.set Task Status (Notice Progress Manager. BPMS\_STATUS\_READY); \\
// 処理ユーザリスト
noticeTask.setUserList(userList):
// 進捗通知タスクをリストにセット
noticeTaskList.add(noticeTask);
            --- プロセス情報を設定する ---
// プロセスのインスタンス生成
BpmsProcess process = new BpmsProcess();
// 業務プロセス名
process.setBusinessProcessName("業務プロセス");
// プロセスインスタンスID
process.setProcessInstanceId(processInstanceId);
// 案件名
process.setProcessName("案件");
```

```
// プロセスステータス。以下の何れかを指定する。
//「未処理」: BPMS_STATUS_READY
//「処理中」: BPMS_STATUS_ACTIVE
//「完了」: BPMS_STATUS_COMPLETE
process.setProcessStatus(NoticeProgressManager.BPMS_STATUS_ACTIVE);
// 進捗通知タスクリスト
process.setNoticeTaskList(noticeTaskList);

try {
    // 進捗通知マネージャの生成
    NoticeProgressManager manager = new NoticeProgressManager("aoyagi", "default");

    // プロセス詳細情報(プロセス・進捗通知タスク・処理ユーザ)の登録
    manager.addBpmsProcess(process);
} catch (BPMSApplicationException e) {
    throw new SystemException(e);
}
```

■ スクリプト開発モデル

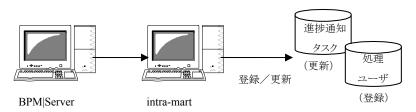
```
// ----- プロセス情報を設定する ------
// 進捗通知タスク詳細オブジェクトの生成
var processDetailInfo = new Object();
// プロセスインスタンス ID
processDetailInfo.processInstanceId = "169875";
// 業務プロセス名
processDetailInfo.businessProcessName = "業務プロセス";
// 案件名
processDetailInfo.processName = "案件":
// プロセスステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」:BPMS STATUS COMPLETE
processDetailInfo.processStatus = NoticeProgressManager.BPMS_STATUS_ACTIVE;
// ----- 進捗通知タスク情報を設定する ------
// 進捗通知タスクオブジェクトの生成
processDetailInfo.noticeTask = new Array(1)::
processDetailInfo.noticeTask[0] = new Object();
// 進捗通知タスク ID
processDetailInfo.noticeTask[0].noticeTaskId = "task1";
// 進捗通知タスク名
processDetailInfo.noticeTask[0].noticeTaskName = "タスク";
// 進捗通知タスクステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」:BPMS_STATUS_COMPLETE
processDetailInfo.noticeTask[0].taskStatus = NoticeProgressManager.BPMS\_STATUS\_READY;
         ----- 処理ユーザ情報を設定する -----
// 処理ユーザオブジェクトの生成
processDetailInfo.noticeTask[0].userList = new Array(1);
processDetailInfo.noticeTask[0].userList[0] = new Object();
// ユーザ CD
processDetailInfo.noticeTask[0].userList[0].userCd = "aoyagi";
// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");
// プロセス詳細情報(プロセス・進捗通知タスク・処理ユーザ)の登録
manager.addBpmsProcess (processDetailInfo);
```

5.2.2 進捗通知タスク情報の登録/更新、処理ユーザ情報の登録

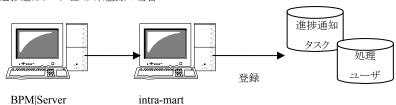
進捗通知タスク情報の登録または更新を行います。また同時にその進捗通知タスクに紐づく処理ユーザの登録も 行います。

このとき、進捗通知タスク ID が登録済の場合は、進捗通知タスク情報の更新と、処理ユーザの登録を行い、進捗通知タスク ID が未登録の場合は、進捗通知タスク情報の登録と、処理ユーザの登録を行います。

<進捗通知タスク ID が登録済の場合>



<進捗通知タスク ID が未登録の場合>



■ JavaEE 開発モデル

```
処理ユーザ情報を設定する -----
// --
// 処理ユーザのリスト
List<BpmsUser> userList = new ArrayList<BpmsUser>();
// 処理ユーザ1
BpmsUser user1 = new BpmsUser();
// ユーザCD
user1.setUserCd("harada");
// 処理ユーザ1をリストにセット
userList.add(user1);
// 処理ユーザ 2
BpmsUser user2 = new BpmsUser();
// ユーザCD
user2.setUserCd("aoyagi");
// 処理ユーザ2をリストにセット
userList.add(user2);
            - 進捗通知タスク情報を設定する -
// 進捗通知タスクのインスタンス生成
NoticeProgressTask noticeTask = new NoticeProgressTask();
// 進捗通知タスク ID
// "task1"が未登録の場合 →進捗通知タスク情報の登録
   "task1"が登録済の場合 →進捗通知タスク情報の更新
noticeTask.setNoticeTaskId("task1");
// 進捗通知タスク名
noticeTask.setNoticeTaskName("タスク名");
// プロセスインスタンスID
noticeTask.setProcessInstanceId("169875");
// 進捗通知タスクステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」: BPMS STATUS COMPLETE
notice Task.set Task Status (Notice Progress Manager.BPMS\_STATUS\_COMPLETE); \\
```

```
// 処理ユーザリスト
noticeTask.setUserList(userList);

// 最終タスクフラグ
boolean endTaskFlg = false;

try {
    // 進捗通知マネージャの生成
    NoticeProgressManager manager = new NoticeProgressManager("aoyagi", "default");

// 進捗通知タスク情報の登録 or 更新と、処理ユーザの登録
    manager.addNoticeTask(noticeTask, endTaskFlg);
} catch (BPMSApplicationException e) {
    throw new SystemException(e);
}
```

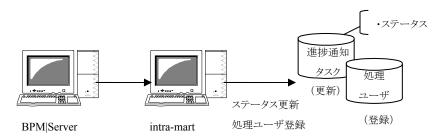
■ スクリプト開発モデル

```
---- 進捗通知タスク情報を設定する ---
// 進捗通知タスク詳細オブジェクトの生成
var noticeTaskDetailInfo = new Object();
// プロセスインスタンスID
noticeTaskDetailInfo.processInstanceId = "169875";
// 進捗通知タスクID
noticeTaskDetailInfo.noticeTaskId = "task1";
// 進捗タスク名
noticeTaskDetailInfo.noticeTaskName = "タスク";
// 進捗タスクステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」:BPMS_STATUS_COMPLETE
noticeTaskDetailInfo.taskStatus = NoticeProgressManager.BPMS_STATUS_COMPLETE;
             -- 処理ユーザ情報を設定する --
// 処理ユーザオブジェクトの生成
noticeTaskDetailInfo.userList = new Array(2);
// 処理ユーザオブジェクト1
noticeTaskDetailInfo.userList[0] = new Object();
// ユーザ CD
noticeTaskDetailInfo.userList[0].userCd = "harada";
// 処理ユーザオブジェクト 2
noticeTaskDetailInfo.userList[1] = new Object();
// ユーザ CD
noticeTaskDetailInfo.userList[1].userCd = "aoyagi";
// 最終タスクフラグ
var endTaskFlg = false;
// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");
// 進捗タスク情報の登録 or 更新と、処理ユーザの登録
manager.addNoticeTask (noticeTaskDetailInfo, endTaskFlg);
```

(※)[進捗一覧-詳細]画面では、**進捗通知タスク ID の昇順**で進捗通知タスク情報を表示しますので、 表示時順を考慮した進捗通知タスク ID を設定してください。 進捗通知タスク ID を省略した場合は一意の値([4 桁の連番]+ユニークな ID)が自動的に振られます。

5.2.3 進捗通知タスクステータスの更新、処理ユーザ情報の登録

登録された進捗通知タスクのステータスを更新し、更新した進捗通知タスクに紐づく処理ユーザ情報の登録を行います。



■ JavaEE 開発モデル

```
// プロセスインスタンス ID
String processInstanceId = "169875";
// 進捗通知タスクID
String noticeTaskId = "task1";
// 進捗通知タスクステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」:BPMS_STATUS_COMPLETE
String taskStatus = NoticeProgressManager.BPMS_STATUS_COMPLETE;
             -- 処理ユーザ情報を設定する -
// 処理ユーザのリスト
List \langle BpmsUser \rangle \ userList = new \ ArrayList \langle BpmsUser \rangle ();
// 処理ユーザ
BpmsUser user = new BpmsUser();
// ユーザCD
user.setUserCd("aoyagi");
// 処理ユーザをリストにセット
userList.add(user);
// 最終タスクフラグ
boolean endTaskFlg = false;
try {
// 進捗通知マネージャの生成
   NoticeProgressManager manager = new NoticeProgressManager("aoyagi", "default");
   // 進捗通知タスクステータスの更新と、処理ユーザの登録
   manager.updateNoticeTaskStatus(processInstanceId, noticeTaskId,
                           taskStatus, userList, endTaskFlg);
} catch (BPMSApplicationException e) {
   throw new SystemException(e);
```

■ スクリプト開発モデル

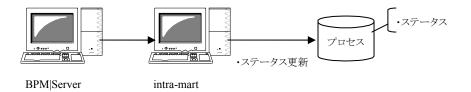
```
// プロセスインスタンス ID
var processInstanceId = "169875";

// 進捗通知タスク ID
var noticeTaskId = "task1";
```

```
// 進捗タスクステータス。以下の何れかを指定する。
//「未処理」:BPMS_STATUS_READY
//「処理中」:BPMS_STATUS_ACTIVE
//「完了」:BPMS_STATUS_COMPLETE
var\ taskStatus = NoticeProgressManager.BPMS\_STATUS\_COMPLETE;
             -- 処理ユーザ情報を設定する ---
// 処理ユーザオブジェクトの生成
userList = new Array(1);
userList[0] = new Object();
// ユーザ CD
userList[0].userCd = "aoyagi";
// 最終タスクフラグ
var endTaskFlg = false;
// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");
// 進捗通知タスクのステータス更新と、処理ユーザ情報の登録
manager.updateNoticeTaskStatus~(processInstanceId,~noticeTaskId,~taskStatus,~userList,~endTaskFlg);\\
```

5.2.4 プロセスステータスの更新

登録されたプロセスのステータスの更新を行います。



■ JavaEE 開発モデル

```
String processInstanceId = "169875";

// プロセスステータス。以下の何れかを指定する。
// 「未処理」: BPMS_STATUS_READY
// 「処理中」: BPMS_STATUS_ACTIVE
// 「完了」: BPMS_STATUS_COMPLETE
String status = NoticeProgressManager. BPMS_STATUS_COMPLETE;

try {
    // 進捗通知マネージャの生成
    NoticeProgressManager = new NoticeProgressManager("aoyagi", "default");

// プロセスステータスの更新
    manager.updateProcessStatus(processInstanceId, status);
} catch (BPMSApplicationException e) {
    throw new SystemException(e);
}
```

■ スクリプト開発モデル

```
// プロセスインスタンス ID
var processInstanceId = "169875";

// プロセスステータス。以下の何れかを指定する。
// 「未処理」: BPMS_STATUS_READY
// 「処理中」: BPMS_STATUS_ACTIVE
// 「完了」: BPMS_STATUS_COMPLETE
var status = NoticeProgressManager.BPMS_STATUS_COMPLETE;

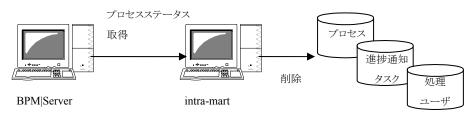
// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");

// プロセスステータスの更新
manager.updateProcessStatus (processInstanceId, status);
```

5.2.5 プロセス詳細情報の削除

プロセスインスタンス ID を指定して、プロセス詳細情報(プロセス・進捗通知タスク・処理ユーザ情報)の削除を行います。削除を実行する前にプロセスのステータスを取得し、削除可能なステータスであるかの判定を行います。

(プロセス詳細情報)



■ JavaEE 開発モデル

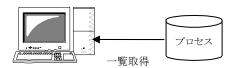
```
// プロセスインスタンス ID
var processInstanceId = "169875";

// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");
// 削除可能なステータスになっているかの判定を取得
var statusComplete = manager.isCompleteProcessInstance(processInstanceId);

// 削除可能なステータスであるかを判定
if (statusComplete.data.isComplete == true){
// プロセス詳細情報の削除
manager.deleteProgress(processInstanceId);
}
```

5.2.6 プロセス一覧の取得

検索条件を指定して、登録されたプロセスの一覧を取得します。



intra-mart

■ JavaEE 開発モデル

```
// ユーザ ID
String userId = "aoyagi";
// 業務プロセス名
String bussinessProcessName = "foo";
// プロセスインスタンス ID →指定なし
String processInstanceId = null,
// 案件名 →指定なし
String processName = null,
// 開始日が"2008/06/1"から"2008/06/30"。
// [yyyy/mm/dd]または[yyyy/mm/dd|24h:mi:ss]で指定する。
String startDateFrom = "2008/06/1";
String startDateTo = "2008/06/30";
// 先頭 1 レコード目から
int startRow = 1;
// 10 件取得
int length = 10;
try {
    // 進捗通知マネージャの生成
    NoticeProgressManager manager = new NoticeProgressManager("aoyagi", "default");
    // ソートを設定 開始日(昇順)
    manager.setProgressSortCondition("start_date asc");
    // プロセス一覧の取得
    BpmsProcess proc = manager.getProcessProgresses(userId, bussinessProcessName,
                          processInstanceId, processName, startDateFrom, startDateTo, startRow, length);
} catch (BPMSApplicationException e) {
    throw new SystemException(e);
```

```
// ユーザ ID
var userld = "aoyagi";
// 業務プロセス名
var bussinessProcessName = "foo";
// プロセスインスタンス ID →指定なし
var processInstanceId= "";
// 案件名 →指定なし
var processName = "";

// 開始日が"2008/06/1"から"2008/06/30"。
// [yyyy/mm/dd]または[yyyy/mm/dd|24h:mi:ss]で指定する。
var startDateFrom = "2008/06/01";
var startDateTo = "2008/06/30";
```

```
// 先頭1レコード目から
var startRow = 1;
// 10 件取得
var length = 10;

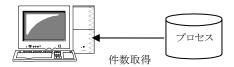
// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");

// ソートを設定 開始日(昇順)
manager.setProgressSortCondition("start_date asc");

// プロセス一覧の取得
var result = manager.getProcessProgresses(userId, bussinessProcessName, processInstanceId, processName, startDateFrom, startDateTo, startRow, length);
var processList = result.data;
```

5.2.7 プロセス件数の取得

検索条件を指定して、登録されたプロセスの件数を取得します。



intra-mart

■ JavaEE 開発モデル

```
// ユーザ ID
String userId = "aoyagi";
// 業務プロセス名
String bussinessProcessName = "foo";
// プロセスインスタンス ID →指定なし
String processInstanceId = null;
// 案件名 →指定なし
String processName = null;
// 開始日が"2008/06/1"から"2008/06/30"。
// [yyyy/mm/dd]または[yyyy/mm/dd|24h:mi:ss]で指定する。
String startDateFrom = "2008/06/01";
String startDateTo = "2008/06/30";
    // 進捗通知マネージャの生成
    NoticeProgressManager manager = new NoticeProgressManager("aoyagi", "default");
    // プロセス件数の取得
    int count = manager.getProcessProgressCount(userId, bussinessProcessName, processInstanceId,
                                            processName, startDateFrom, startDateTo);
} catch (BPMSApplicationException e) {
    throw new SystemException(e);
```

```
// ユーザ ID
var userId = "aoyagi";
// 業務プロセス名
var bussinessProcessName = "foo";
// プロセスインスタンス ID →指定なし
var processInstanceId= "";
// 案件名 →指定なし
var processName = "";
// 開始日が"2008/06/1"から"2008/06/30"。
// [yyyy/mm/dd]または[yyyy/mm/dd|24h:mi:ss]で指定する。
var startDateFrom = "2008/06/01";
var startDateTo = "2008/06/30";
// 進捗通知マネージャの生成
var manager = new NoticeProgressManager("aoyagi", "default");
// プロセス件数の取得
var result = manager. getProcessProgressCount(userId, bussinessProcessName, processInstanceId,
processName, startDateFrom, startDateTo);
var processCount = result.data.count;
```

5.3 TaskManager

5.3.1 タスクの取得

指定したタスクIDのタスクを取得します。タスクマネージャ生成時に指定したユーザIDに関するタスクのみを取得します。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関するタスクの取得
Task task = manager.getTask(taskId);
```

■ スクリプト開発モデル

```
// タスクID
var taskId = "task_id";

// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");

// タスクの取得
var result = taskManager.getTask(taskId);
var task = result.data;
```

5.3.2 タスクー覧の取得

■ JavaEE 開発モデル

タスク一覧を全件取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみを取得します。

```
// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関するタスク一覧の取得
Task[] tasks = manager.getTaskList();
```

検索条件を指定してタスク一覧を取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみを取得します。

```
// 検索条件
TaskCondition condition = new TaskCondition();
// 案件に「foo」が含まれる
condition.descriptionContains("foo");
// タスク種別:「処理」、「通知」
condition.taskTypeIn(TaskType.ACTIVITY, TaskType.NOTIFICATION);
// タスク状態:「通常」、「保留」
condition.taskStateIn(TaskState.READY, TaskState.CLAIMED);
// 作成日(昇順)、案件名(降順)
condition.orderBy(TaskColumn.CREATION_DATE, true);
condition.orderBy(TaskColumn.DESCRIPTION, false);

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関するタスク一覧の取得
Task[] tasks = manager.getTaskList(condition);
```

■ スクリプト開発モデル

検索条件を指定してタスク一覧を取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみを取得します。

```
// 検索条件
var condition = {
   // タスク種別:「処理」、「通知」
   taskTypeIn: [TaskType.ACTIVITY, TaskType.NOTIFICATION],
   // タスク状態:「通常」、「保留」
   taskStateIn: [TaskState.READY, TaskState.CLAIMED],
   // 案件名に"foo"が含まれる
   descriptionContains: "foo",
   // 作成日(昇順), 案件名(降順)
   order By: [Task Column. CREATION\_DATE\_ASC, Task Column. DESCRIPTION\_DESC], \\
   // 先頭1レコード目から
   offset: 1,
   // 10 件取得
   count: 10
// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");
// ユーザ"aoyagi"に関するタスク一覧の取得
var result = taskManager.getTaskList(condition);
var tasks = result.data;
```

タスク件数の取得 5.3.3

■ JavaEE 開発モデル

タスクの件数を取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみをカウントします。

```
// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");
// ユーザ"aoyagi"に関するタスク件数の取得
int count = manager.getTaskCount();
```

検索条件を指定して、タスク件数を取得します。タスクマネージャ生成時に指定したユーザIDに関するタスクのみ をカウントします。

```
// 検索条件
TaskCondition condition = new TaskCondition();
// タスク種別:「処理」
condition.taskTypeIn(TaskType.ACTIVITY);
// タスク状態:「通常」、「保留」
condition.taskStateIn(TaskState.READY, TaskState.CLAIMED);
// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");
// ユーザ"aoyagi"に関するタスク件数の取得
int count = manager.getTaskCount(condition);
```

■ スクリプト開発モデル

検索条件を指定して、タスク件数を取得します。タスクマネージャ生成時に指定したユーザIDに関するタスクのみ をカウントします。

```
// 検索条件
var condition = {
// タスク種別:「処理」
   taskTypeIn: [TaskType.ACTIVITY],
    // タスク状態:「通常」、「保留」
   taskStateIn: [TaskState.READY, TaskState.CLAIMED]
// タスクマネージャの牛成
var taskManager = new TaskManager("aoyagi", "default");
// ユーザ"aoyagi"に関するタスク件数の取得
var result = taskManager.getTaskCount(condition);
var count = result.data;
```

起票タスク一覧の取得 5.3.4

起票タスク一覧を取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみを取得します。

■ JavaEE 開発モデル

```
// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");
// ユーザ"aoyagi"に関する起票タスク一覧の取得
InitialTask[] tasks = manager.getInitialTaskList();
```

■ スクリプト開発モデル

```
// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関する起票タスク一覧の取得
var result = taskManager.getInitialTaskList();
var initialTasks = result.data;
```

5.3.5 処理タスク一覧の取得

処理タスク一覧を取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみを取得します。

■ JavaEE 開発モデル

```
// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関する処理タスク一覧の取得
ActivityTask□ tasks = manager.getActivityTaskList();
```

■ スクリプト開発モデル

```
// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関する処理タスク一覧の取得
var result = taskManager.getActivityTaskList();
var activityTasks = result.data;
```

5.3.6 通知タスク一覧の取得

通知タスク一覧を取得します。タスクマネージャ生成時に指定したユーザ ID に関するタスクのみを取得します。

■ JavaEE 開発モデル

```
// タスクマネージャの生成
TaskManager manager = new TaskManager(″aoyagi″, ″default″);

// ユーザ″aoyagi″に関する通知タスク一覧の取得
NotificationTask□ tasks = manager.getNotificationTaskList();
```

```
// タスクマネージャの生成
var taskManager = new TaskManager (″aoyagi″, ″default″);

// ユーザ″aoyagi″に関する通知タスク一覧の取得
var result = taskManager.getNotificationTaskList();
var notificationTasks = result.data;
```

5.3.7 処理済タスク一覧の取得

処理済タスク一覧を取得します。タスクマネージャ生成時に指定したユーザIDに関するタスクのみを取得します。

■ JavaEE 開発モデル

```
// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// ユーザ"aoyagi"に関する処理済タスク一覧の取得
Task[] tasks = manager.getCompletedTaskList();
```

■ スクリプト開発モデル

```
// タスクマネージャの生成
var taskManager = new TaskManager (″aoyagi″, ″default″);

// ユーザ″aoyagi″に関する処理済タスク一覧の取得
var result = taskManager.getCompletedTaskList();
var completedTasks = result.data;
```

5.3.8 プロセスの開始

タスク ID を指定して、プロセスの開始を行います。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";
// 出力ドキュメント
Document doc = XMLConversionUtils.convertToDocument("<input xmlns=\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://...\footnote{"http://
```

```
// タスク ID
var taskId = "task_id";
// 出力ドキュメント
var output = <input xmlns="http://..." />;

// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");

// プロセスの開始
taskManager.init(taskId, output);
```

5.3.9 処理タスクの完了

タスク ID を指定して、処理タスクの完了を行います。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";
// 出力ドキュメント
Document doc = XMLConversionUtils.convertToDocument("<input xmlns=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotation=\footnotati
```

■ スクリプト開発モデル

```
// タスク ID
var taskId = "task_id";
// 出力ドキュメント
var output = <input xmlns="http://..." />;

// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");

// 処理タスクの完了
taskManager.complete(taskId, output);
```

5.3.10 処理タスクの保留

タスク ID を指定して、処理タスクの保留を行います。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// 処理タスクの保留
manager.claimTask(taskId);
```

```
// タスク ID
var taskId = "task_id";

// タスクマネージャの生成
var taskManager = new TaskManager("aoyagi", "default");

// 処理タスクの保留
taskManager.claim(taskId);
```

5.3.11 処理タスクの保留→完了

タスク ID を指定して、処理タスクを保留した後、完了を行います。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";

// 出力ドキュメント
Document doc = XMLConversionUtils.convertToDocument("⟨input xmlns=¥"http://...¥" />");

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// 処理タスクの保留→完了
manager.claimAndCompleteTask(taskId, doc);
```

■ スクリプト開発モデル

```
// タスク ID
var taskId = "task_id";

// 出力ドキュメント
var output = ⟨input xmlns="http://..." /〉;

// タスクマネージャの生成
var taskManager = new TaskManager ("aoyagi", "default");

// 処理タスクの保留→完了
taskManager.claimAndComplete(taskId, output);
```

5.3.12 処理タスクの保留取消

タスク ID を指定して、処理タスクの保留取消を行います。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// 処理タスクの保留取消
manager.revokeTask(taskId);
```

```
// タスク ID
var taskId = "task_id";

// タスクマネージャの生成
var taskManager = new TaskManager ("aoyagi", "default");

// 処理タスクの保留取り消し
taskManager.revoke(taskId);
```

5.3.13 処理タスクの一時保存

タスク ID を指定して、処理タスクの一時保存を行います。

■ JavaEE 開発モデル

```
// タスク ID
String taskId = "task_id";
// 出力ドキュメント
Document doc = XMLConversionUtils.convertToDocument("<input xmlns=\text{"http://...\text{\text{\text{\text{input xmlns}}}" // ** />");

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// 処理タスクの一時保存
manager.saveTask(taskId, doc);
```

■ スクリプト開発モデル

```
// タスクID
var taskId = "task_id";
// 出力ドキュメント
var output = <input xmlns="http://..." />;
// タスクマネージャの生成
var taskManager = new TaskManager ("aoyagi", "default");
// 処理タスクの一時保存
taskManager.save(taskId, output);
```

5.3.14 通知タスクの確認

タスク ID を指定して、通知タスクの確認を行います。

■ JavaEE 開発モデル

```
// 通知タスクのタスク ID
String taskId = "task_id";

// タスクマネージャの生成
TaskManager manager = new TaskManager("aoyagi", "default");

// 通知タスクの確認
manager.dismissTask(taskId);
```

```
// タスク ID
var taskId = "task_id";

// タスクマネージャの生成
var taskManager = new TaskManager ("aoyagi", "default");

// 通知タスクの確認
taskManager.dismiss(taskId);
```

5.4 TokenManager

5.4.1 トークンの生成

トークンの生成を行います。

■ JavaEE 開発モデル

```
// トークンマネージャの生成
TokenManager manager = new TokenManager ("default");

// トークンの生成
String token = manager.createToken("aoyagi");
```

■ スクリプト開発モデル

```
//トークンマネージャの生成
var tokenManager = new TokenManager("default");

//トークンの生成
var result = tokenManager.createToken("aoyagi");
var participantToken = result.data;
```

5.4.2 ユーザIDの変換

■ JavaEE 開発モデル

```
//トークンマネージャの生成
TokenManager manager = new TokenManager ("default");
//「グループID+"¥"+ ユーザ ID」形式の文字列に変換
String user = getBPMSUser("aoyagi");
```

```
//トークンマネージャの生成
var tokenManager = new TokenManager("default");

//「グループ ID + "¥" + ユーザ ID」形式の文字列に変換。
var result = tokenManager.getBPMSUser ("aoyagi");
var bpmsUser = result.data;
```

5.4.3 ロールー覧の取得

BPMS サーバ上で利用されるロール一覧を取得します。

■ JavaEE 開発モデル

```
// トークンマネージャの生成
TokenManager manager = new TokenManager (″default″);

// ロール一覧の取得
String□ roles = manager.getUserRoles(″aoyagi″);
```

5.4.4 トークンの解析

トークンの解析を行います。

■ JavaEE 開発モデル

```
String token = ".....";

//トークンマネージャの生成
TokenManager manager = new TokenManager ("default");

//トークンの解析
Properties properties = manager.parseToken(token);

// 解析結果取得
String issued = properties.getProperty(TokenManager.ISSUED);
String user = properties.getProperty(TokenManager.USER);
String roles = properties.getProperty(TokenManager.ROLES);
String fullName = properties.getProperty(TokenManager.FULLNAME);
```

intra-mart WebPlatform/AppFramework Ver. 7. 0 BPM プログラミングガイド

2008/08/22 初版

Copyright 2000-2008 株式会社 NTT データ イントラマート

All rights Reserved.

TEL: 03-5549-2821 FAX: 03-5549-2816

E-MAIL: info@intra-mart.jp URL: http://www.intra-mart.jp/