Implementing Inter-Process Communication in IBM Business Automation Manager Open Edition (IBM BAMOE)

Author: Bob Riaz (sriaz@salientprocess.com)

Contents

Introduction

If a process embodies a piece of reusable logic it would save development effort if this process could be invoked from other processes that require that very piece of logic. For instance, a Loan Application project and a Loan Servicing project might both have a need for a user task called Update Loan Params in which the Loan information is modified. A reusable process which manages this task would serve the needs of both projects. In IBM BAMOE a process is defined within a project. In a given workspace there may be numerous projects with processes defined within each.

This document attempts to describe the various ways in which one process in IBM BAMOE may be invoked from another; as data mapping from one process to the other differs depending on the method of invocation code snippets have been provided to explain how to map and manage data in interprocess communication. For the most part the code contained may be used as is with modifications to suit your specific needs, but in general, it would be a good idea to write your script logic in an IDE such as Eclipse to ensure the code compiles. Also, since the code inside an action script is code that would normally be contained in a Java method there seems to be no way to add “import” statements, which means that all references to objects must be fully qualified, with a few exceptions such as String.

Communication Between Processes in Different Projects

Example scenario:

- ➔ A Loan application project in which a loan is entered in one task (Enter Loan Application task) and updated in another task (Update Loan Params).
- ➔ The Enter Loan Application task is defined in one project (Loan Project) the Update Loan Params task is defined in a different project (Loan Reusable Services).

This can be achieved in two ways

1. *By including the reusable project as a dependency in the parent project*
2. *By invoking the reusable service via a REST call from the parent project*

By Including One Project as a Dependency in Another Project:

Notes: The steps below are partially based on a 2014 community article: Reuse your business assets with jBPM 6.

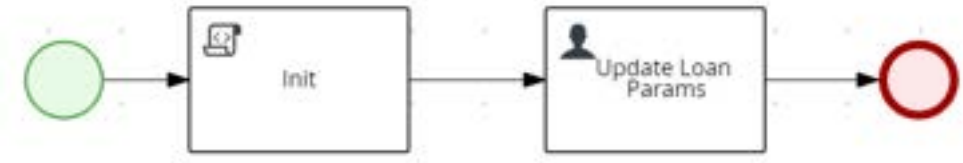
We've found that including one project as a dependency in another via KIE bases can produce unreliable results as the number of dependencies increases. We are including this method here for completeness, but in general, we would recommend using one of the other methods of inter-process communication described below.

In the definition of the process we would like to be able to include the Update Loan Params task/service as part of the flow of the service in which the Enter Loan Application task is defined making the two tasks flow synchronously.

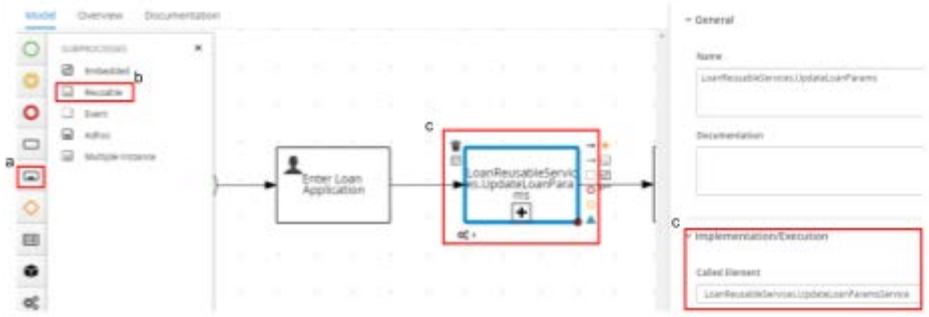
1. Create the Loan Project
2. Create the Loan Reusable Services project
3. Loan Reusable Services project, open the Settings tab
    a. Select KIE bases
    b. Click Add KIE base
    c. The first entry that appears will have Default selected
    d. Click Add KIE base again
    e. The second entry that now appears will not have Default selected. This is the one we want. Delete the entry with Default selected.
    f. Enter a name for the KIE base for the project. This name will need to be unique across all projects
    g. Under Packages enter "*" to include all packages
    h. Click the KIE sessions link

i. In the dialog box, click Add KIE session
j. The first entry that appears will have Default selected
k. Click Add KIE session again
l. The second entry that now appears will not have Default selected. This is the one we want. Delete the entry with Default selected.
m. Enter a name for the KIE session for the project. This name will need to be unique across all projects
n. Under State enter "stateful"
o. Click Done

4. Back in Settings click Save
5. Create a reusable process. Eg. Update Loan Params. Each project will manage its own dependencies. In this example the parent process will map a Loan object into the reusable process. This means that both projects will need to have the requisite dependency to reference the Loan object.

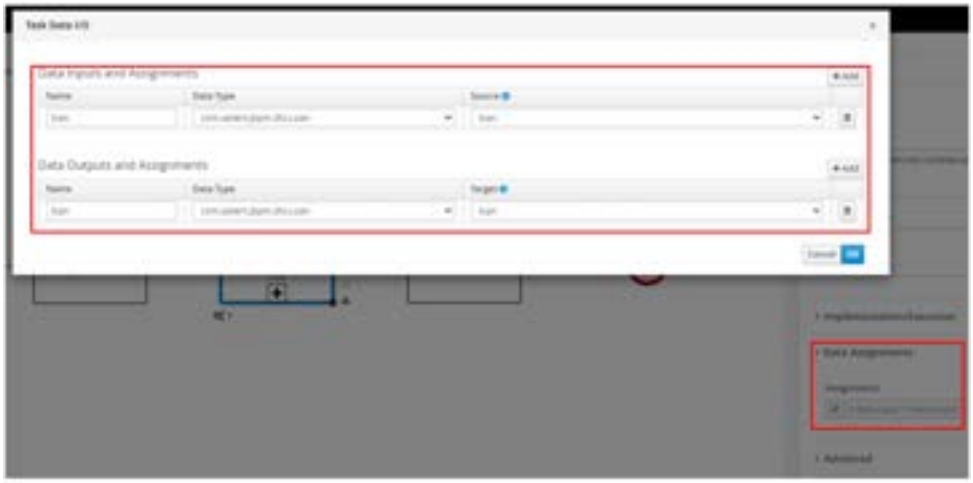


6. Build and deploy the project
7. There will be an error in the logs but apparently this can be ignored
8. In the Loan Project, create a process that uses the re-usable process created above.
    a. In the process editor, select SubProcesses from the left nav
    b. Select the Reusable artifact and drag it onto the canvas
    c. With the Reusable artifact selected in the canvas select the appropriate Called Element under Implementation



9. The finished parent process might look like this.

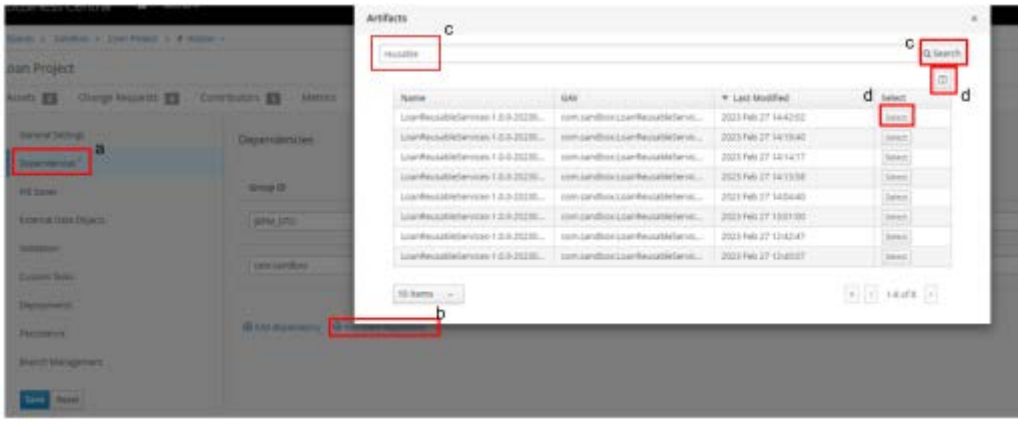
10. Map data variables into the Reusable service via the Data Assignments dialog box



11. In the parent project open the Settings tab
    a. Select KIE bases
    b. Click Add KIE base
    c. The first entry that appears will have Default selected. This is the entry we want.
    d. Enter a name for the KIE base for the project. This name will need to be unique across all projects
    e. Under Included KIE Bases click Add
    f. Enter the name of the KIE Base given in the reusable project above
    g. Under Packages enter * to include all packages
    h. Click the KIE sessions link

i. In the dialog box, click Add KIE session
j. The first entry that appears will have Default selected. This is the entry we want.
k. Enter a name for the KIE session for the project. This name will need to be unique across all projects
l. Under State enter "stateful"
m. Click Done
n. Back in Settings click Save

12. Add the reusable project as a dependency
    a. Click Dependencies
    b. Click Add from Repository
    c. In the dialog box enter part of the project name and Search for the project
    d. Add the Last Modified column and select the latest version of the project



e. Back in Settings click Save

By Invoking a Service Defined in One Project From a Service Defined in a Different Project using REST

(i.e. using the KIE REST API)

Example:

- ➔ The caller project is Mortgage Sales. This project has a service MortgageApplicationIntake. From this project we wish to invoke a service defined in some other project.
- ➔ The project is called Mortgage Fulfillment. This project has a serviceMortgageApplicationApproval.

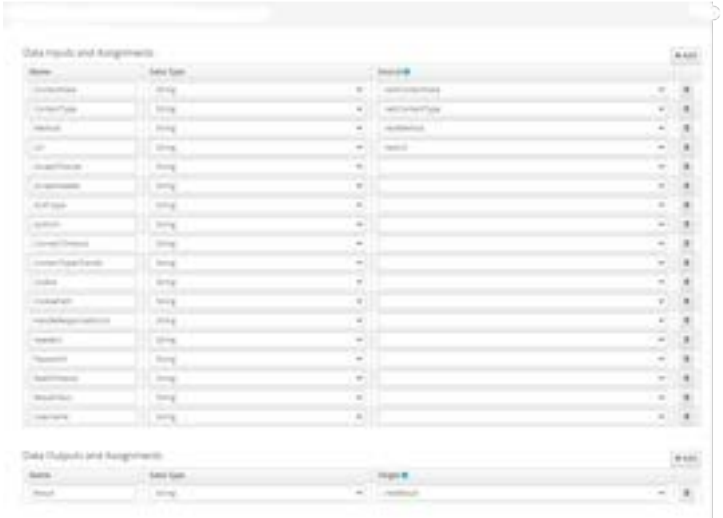
1. Set a service to make a REST call (Setting up REST)
2. There are two KIE REST API's that can be used to start a process:
   i. /server/containers/{containerId}/processes/{processId}/instances
   Starts a process asynchronously, return value is the process instance id of the process instance started
   ii. /server/containers/{containerId}/processes/{processId}/computedInstances
   Starts a process asynchronously, return values are any values set in the kcontext of the process started. The return value comes back in JSON format.

3. `/server/containers/MortgageFulfilment_1.0.0-SNAPSHOT/processes/MortgageFulfilment.MortgageApplicationApproval/instances` – This API will start MortgageFulfilment.MortgageApplicationApproval process and return the process instance id.
   To set up complex data as the payload of a REST call
4. Create a JSON string containing the variables initialized. You can use the Gson library (ships with BAMOE) to convert between Java and JSON. Confirm the gson jar exists in …\standalone\deployments\kie-server.war\WEB-INF\lib and … \standalone\deployments\business-central.war\WEB-INF\lib
   The artifact is also available at https://mvnrepository.com/artifact/com.google.code.gson
   i.

```
Eg.
processInstanceId = (Long) kcontext.getProcessInstance().getId();

loan = new com.salient.jbpm.dto.Loan();
loan.setLoanNumber("1234");
com.salient.jbpm.dto.Person applicant = new com.salient.jbpm.dto.Person();
applicant.setFirstName("Thelonious");
applicant.setLastName("Monk");
loan.setApplicant(applicant);
loan.setLoanAmount(new Double(10000));
callbackSignal = "updateLoanCompleted_" + processInstanceId;

java.util.Map<String, Object> variablesMap = new java.util.HashMap<String, Object>();
java.util.Map<String, Object> loanMap = new java.util.HashMap<String, Object>();
variablesMap.put("name", "Thelonious Monk");
loanMap.put("com.salient.jbpm.dto.Loan", loan);
variablesMap.put("loan", loanMap);
variablesMap.put("callbackSignal", callbackSignal);

//Use the Gson library to create a JSON string out of the map
com.google.gson.Gson gson = new com.google.gson.Gson();
restContentData = gson.toJson(variablesMap);
/*
```

```
This will produce a JSON string:
{
    "callbackSignal": "updateLoanCompleted_1545",
    "loan": {
        "com.salient.jbpm.dto.Loan": {
            "loanNumber": "1234",
            "applicant": {
                "firstName": "Thelonious",
                "lastName": "Monk"
            },
            "loanAmount": 10000.0
        }
    },
    "name": "Thelonious Monk"
}

The class information (com.salient.jbpm.dto.Loan) of the complex object property is added
as a field of the JSON string to facilitate parsing data sent via a REST API. In the
absence of class information for a structure in a JSON string, JBPM converts the structure
into a LinkedHashMap. When the class information is supplied JBPM will return an instance
of the class rather than
a LinkedHashMap. Eg. If the JSON above is sent via a REST call to a service,
you can access the loan object directly from the kcontext with
loan = (com.salient.jbpm.dto.Loan) kcontext.getVariable("loan");
Without the class information getting tha loan object would involve getting the
LinkedHashMap first,
java.util.LinkedHashMap loanMap = (java.util.LinkedHashMap) kcontext.getVariable("loan");
and then converting the LinkedHashMap object to its actual type.
*/
//Set the other REST work item handler inputs
restUrl = "http://localhost:8080/kie-server/services/rest/server/containers/MortgageFulfil
ment_1.0.0-SNAPSHOT/processes/MortgageFulfilment.MortgageApplicationApproval/instances";

restContentType = "application/json";
restMethod = "POST";


//Set all variables to the kcontext to make them available to be mapped into the
//REST work item handler
kcontext.setVariable("restUrl", restUrl);
kcontext.setVariable("restContentData", restContentData);
kcontext.setVariable("restContentType", restContentType);
kcontext.setVariable("restMethod", restMethod);
```

ii. Map the inputs to the REST work item handler and execute the process.



iii. The process instance id of the new process will be available in the restResult variable mapped from the output of the REST API.

5. The MortgageFulfilment.MortgageApplicationApproval simply extracts the data from the REST call, and updates two values in the loan object.

```
//Expecting: loan, callbackSignal, name
System.out.println("MortgageApplicationApproval Parse input data...");
if (kcontext.getVariable("loan") != null) {
    loan = (com.salient.jbpm.dto.Loan) kcontext.getVariable("loan");
    System.out.println("MortgageApplicationApproval loan.LoanNumber:::" + loan.getLoanNumber());
}
callbackSignal = (String) kcontext.getVariable("callbackSignal");

//Update loan values
loan.setLoanAmount(new Double(25500));
loan.setStatus("APPROVED");
kcontext.setVariable("loan", loan);
```

6. When a JSON string such as the one in the example above, which contains three properties (name, loan, and callbackSignal) is set up as the payload to the called process via the REST API, BAMOE makes the properties of the REST payload available to the receiver in the kcontext. Thus, it's possible to extract individual properties of the payload directly:
`loan = (com.salient.jbpm.dto.Loan) kcontext.getVariable("loan");`

   But if you need to parse a JSON string directly, below are the steps:
   Eg. String `variablesJSON` is set to the JSON string below

```
{
  "callbackSignal": "updateLoanCompleted_1545",
  "loan": {
          "com.salient.jbpm.dto.Loan": {
                "loanNumber": "1234",
                "applicant": {
                  "firstName": "Thelonious",
                  "lastName": "Monk"
                },
                "loanAmount": 10000.0
              }
           },
  "name": "Thelonious Monk"
}

//A simple variable can be extracted directly via the kcontext
String callbackSignal = (String) kcontext.getVariable("callbackSignal");

//A complex variable such as the loan object will be available as a map
com.google.gson.Gson gson = new com.google.gson.Gson();

//Convert JSON into Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> variablesMap = (com.google.gson.internal.Li
nkedTreeMap<String, Object>) gson.fromJson(variablesJSON, java.util.Map.class);

//If map value is simple get the value directly
String name = (String) variablesMap.get("name");

//If map value is complex get the object as a Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> loanMap = (com.google.gson.internal.LinkedT
reeMap<String, Object>) variablesMap.get("loan");

//Get the values of the complex object as a Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> loanValuesMap =  (com.google.gson.internal.
LinkedTreeMap<String, Object>) loanMap.get("com.salient.jbpm.dto.Loan");
```

```
//Convert the complex object values map into a JSON string
String loanValuesJSON = gson.toJson(loanValuesMap);

//Convert the complex object values map JSON string into the actual object
com.salient.jbpm.dto.Loan loanObject = gson.fromJson(loanValuesJSON, com.salient.jbpm.dto.Loan.cla
ss);
//Gson recognizes classes that reside in project dependencies. Eg.com.salient.jbpm.dto.Loan
//lives in the project dependency JBPM_DTO
```

See steps to add an artifact as a dependency to a project

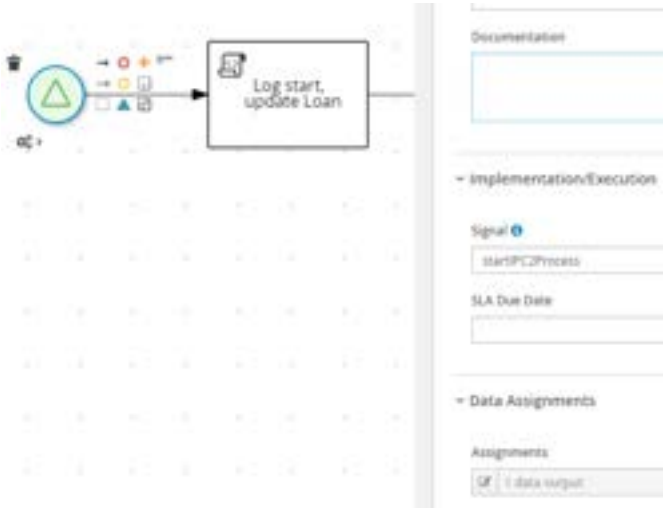
7. /server/containers/{containerId}/processes/{processId}/computedInstances – This API will start a process synchronously. Data saved to the kcontext in the child process will be available in the restResult in JSON format. Follow the steps to set up the data to send via the REST call. We will call the same process, MortgageFulfilment.MortgageApplicationApproval but via the synchronous REST API. This time the updated loan will be available in the restResult.

```
{
  "processInstanceId": 1552,
  "callbackSignal": "updateLoanCompleted_1552",
  "loan": {
    "com.salient.jbpm.dto.Loan": {
      "loanNumber": "1234",
      "loanType": null,
      "applicant": {
        "com.salient.jbpm.dto.Person": {
          "id": null,
          "firstName": "Thelonious",
          "lastName": "Monk",
          "ssn": null,
          "income": null,
          "ficoScore": null
        }
      },
      "loanAmount": 25500.0,
      "status": "APPROVED"
    },
    "containerId": "MortgageSales_1.0.0-SNAPSHOT",
    "initiator": "pamAdmin"
  }
}
```

By Invoking a Service Defined in One Project From a Service Defined in a Different Project using JMS

Set up JMS

In this example we will be starting a process that starts with a Start Signal.



The payload is mapped directly into a variable.

To set up the payload to send via JMS:

```
//Get current processInstanceId
processInstanceId = (Long) kcontext.getProcessInstance().getId();
//Set callback signal if necessary
callbackSignal = "IPC2ProcessComplete";

//Initialize complex object to be sent in payload
loan = new com.salient.jbpm.dto.Loan();
loan.setLoanNumber("1234");
com.salient.jbpm.dto.Person applicant = new com.salient.jbpm.dto.Person();
applicant.setFirstName("Thelonious");
applicant.setLastName("Monk");
loan.setApplicant(applicant);
loan.setLoanAmount(new Double(10000));
kcontext.setVariable("loan", loan);

//Create Map to set the variables in
java.util.Map<String, Object> variablesMap = new java.util.HashMap<String, Object>();
variablesMap.put("callerProcessInstanceId", processInstanceId);
java.util.Map<String, Object> loanMap = new java.util.HashMap<String, Object>();
loanMap.put("com.salient.jbpm.dto.Loan", loan);

variablesMap.put("loan", loanMap);
variablesMap.put("callbackSignal", callbackSignal);
```

```
variablesMap.put("callerProcessInstanceId", processInstanceId);

com.google.gson.Gson gson = new com.google.gson.Gson();
messagePayload = gson.toJson(variablesMap);

//The resulting message payload will look like this
/*
{
    "callbackSignal": "IPC2ProcessComplete",
    "loan": {
        "com.salient.jbpm.dto.Loan": {
            "loanNumber": "1234",
            "applicant": {
                "firstName": "Thelonious",
                "lastName": "Monk"
            },
            "loanAmount": 10000.0
        }
    },
    "callerProcessInstanceId": 43
}
*/

targetDeploymentId = "InterProcessCommunication2_1.0.0-SNAPSHOT";

signalMessage = "startIPC2Process";

kcontext.setVariable("signalMessage", signalMessage);
kcontext.setVariable("targetDeploymentId", targetDeploymentId);
kcontext.setVariable("messagePayload", messagePayload);
kcontext.setVariable("callbackSignal", callbackSignal);
```

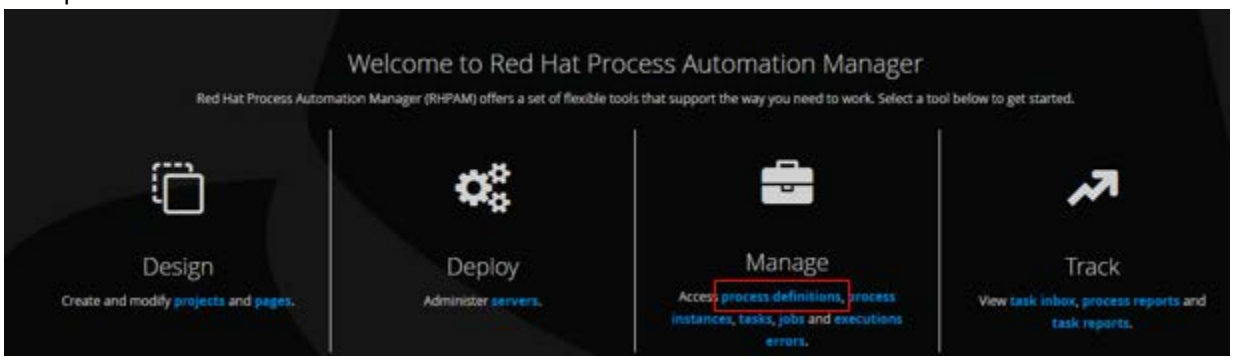
When the payload is received it can be accessed directly from the variable into which it was mapped.

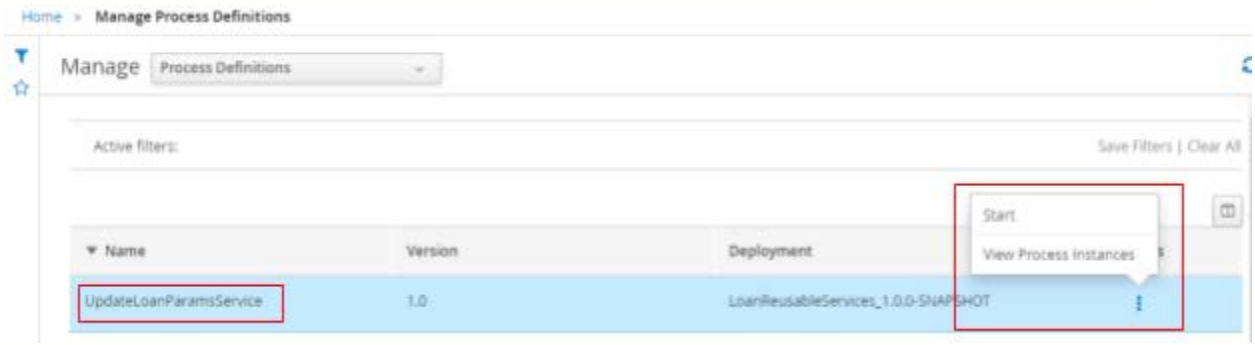
```
messagePayload = (String) kcontext.getVariable("messagePayload");
System.out.println("IPC2 messagePayload received:::" + messagePayload);

com.google.gson.Gson gson = new com.google.gson.Gson();
//Convert JSON into Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> variablesMap = (com.google.gson.internal.LinkedTree
Map<String, Object>) gson.fromJson(messagePayload, java.util.Map.class);

//If map value is simple get the value directly
callbackSignal = (String) variablesMap.get("callbackSignal");
callerProcessInstanceId = ((Double)variablesMap.get("callerProcessInstanceId")).longValue();
callerDeploymentId = (String) variablesMap.get("callerDeploymentId");

//If map value is complex get the object as a Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> loanMap = (com.google.gson.internal.LinkedTreeMap<S
tring, Object>) variablesMap.get("loan");

//Get the values of the complex object as a Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> loanValuesMap =  (com.google.gson.internal.LinkedTr
eeMap<String, Object>) loanMap.get("com.salient.jbpm.dto.Loan");

//Convert the complex object values map into a JSON string
String loanValuesJSON = gson.toJson(loanValuesMap);
System.out.println("IPC2 loan received:::" + loanValuesJSON);

//Convert the complex object values map JSON string into the actual object
loan = gson.fromJson(loanValuesJSON, com.salient.jbpm.dto.Loan.class);
loan.setLoanType("AUTO");
loan.setLoanAmount(new Double(150000));
loan.setStatus("APPROVED");

kcontext.setVariable("callbackSignal", callbackSignal);
kcontext.setVariable("loan", loan);
```

Communication Between Processes Defined in the Same Project

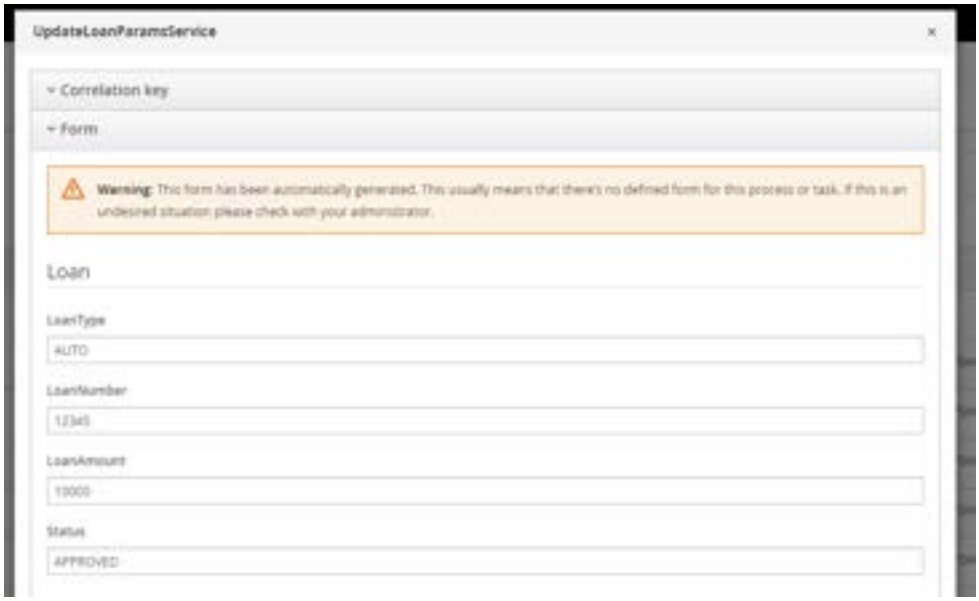
To start a process that starts with a None Start Event:

1. Start process via Process Definitions





Payload:
Enter values for process variables in default form



2. Start process without parameters

```
org.kie.api.runtime.KieSession ksession = kcontext.getKieRuntime().getKieBase().newKieSession();
org.kie.api.runtime.process.ProcessInstance processInstance =
ksession.startProcess("{containerId}.{processId}");
```

3. Start process with parameters

```
org.kie.api.runtime.KieSession ksession = kcontext.getKieRuntime().getKieBase().newKieSession();
org.kie.api.runtime.process.ProcessInstance processInstance =
ksession.startProcess("InterProcessCommunication1.ProcessB", params);
```

Payload:

```
//Send
java.util.Map params = new java.util.HashMap<String, Object>();
loan = new com.salient.jbpm.dto.Loan();
loan.setLoanNumber("1234");
com.salient.jbpm.dto.Person applicant = new com.salient.jbpm.dto.Person();
applicant.setFirstName("Thelonious");
applicant.setLastName("Monk");
loan.setApplicant(applicant);
loan.setLoanAmount(new Double(10000));
params.put("loan", loan);

//Receive
loan = (com.salient.jbpm.dto.Loan)kcontext.getVariable("loan");
```

Issues:
If the process being started via the API has a human task an exception is thrown: Could not find work item handler for Human Task

4. Start as a Reusable process
   Payload:
   Map variables from the parent to the Reusable process via Data Assignments

To start a process that starts with a Signal Start Event:

1. Use a Throw Intermediate Signal event to start a process that has a Signal Start Event. The sent and received Signals must match.
   Payload:
   The Throw Intermediate Signal event Data Assignments allows only one variable to be mapped. If multiple variables must be passed to the child process use a Map variable or a JSON string.
   Example using a Map variable:

```
//Send signal event set up
loan = new com.salient.jbpm.dto.Loan();
loan.setLoanNumber("1234");
com.salient.jbpm.dto.Person applicant = new com.salient.jbpm.dto.Person();
applicant.setFirstName("Thelonious");
applicant.setLastName("Monk");
loan.setApplicant(applicant);
loan.setLoanAmount(new Double(10000));
kcontext.setVariable("loan", loan);
processInstanceId = (Long) kcontext.getProcessInstance().getId();
signalCode = "startSignalProcessC"; //"startSignalProcess";
callbackSignalCode = "signalProcessCompleted_" + processInstanceId;
kcontext.setVariable("signalCode", signalCode);
kcontext.setVariable("callbackSignalCode", callbackSignalCode);
params = new java.util.HashMap<String, Object>();
params.put("loan", loan);
params.put("processInstanceId", processInstanceId);
params.put("callbackSignalCode", callbackSignalCode);
kcontext.setVariable("params", params);

//Receive signal event set up
```

```
//Inputs from the caller will not be available in the kcontext, get them from the params of the
//signal event
params = (java.util.HashMap)kcontext.getVariable("params");
loan = (com.salient.jbpm.dto.Loan)params.get("loan");
System.out.println("ProcessCWithSignalStart LoanNumber:::" + loan.getLoanNumber());
System.out.println("ProcessCWithSignalStart callerProcessInstanceId:::" +
callerProcessInstanceId);
signalCode = (String)params.get("callbackSignalCode");
System.out.println("ProcessCWithSignalStart signalCode:::" + signalCode);
loan.setLoanNumber("987654");
kcontext.setVariable("loan", loan);

kcontext.setVariable("signalCode", signalCode);
System.out.println("ProcessCWithSignalStart signalCode:::" + signalCode);
```

Example using a JSON string:

```
//Send
//Use a JSON string as signal payload
java.util.Map<String, Object> variablesMap = new java.util.HashMap<String, Object>();
java.util.Map<String, Object> loanMap = new java.util.HashMap<String, Object>();
variablesMap.put("processInstanceId", processInstanceId);
variablesMap.put("callbackSignalCode", callbackSignalCode);
loanMap.put("com.salient.jbpm.dto.Loan", loan);
variablesMap.put("loan", loanMap);
//Use the Gson library to create a JSON string out of the map
com.google.gson.Gson gson = new com.google.gson.Gson();
String paramsJSON = gson.toJson(variablesMap);

System.out.println("paramsJSON:::" + paramsJSON);

//Receive
//Inputs from the caller will not be available in the kcontext, get them from the paramsJSON of
//the signal event
paramsJSON = (String) kcontext.getVariable("paramsJSON");

com.google.gson.Gson gson = new com.google.gson.Gson();
//Convert JSON into Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> variablesMap =
(com.google.gson.internal.LinkedTreeMap<String, Object>) gson.fromJson(paramsJSON,
java.util.Map.class);
//If map value is simple get the value directly
callerProcessInstanceId = ((Double)variablesMap.get("processInstanceId")).longValue();
signalCode = (String) variablesMap.get("callbackSignalCode")
//If map value is complex get the object as a Map
com.google.gson.internal.LinkedTreeMap<String, Object> loanMap =
(com.google.gson.internal.LinkedTreeMap<String, Object>) variablesMap.get("loan");
//Get the values of the complex object as a Map (com.google.gson.internal.LinkedTreeMap)
com.google.gson.internal.LinkedTreeMap<String, Object> loanValuesMap =
(com.google.gson.internal.LinkedTreeMap<String, Object>) loanMap.get("com.salient.jbpm.dto.Loan");
//Convert the complex object values map into a JSON string
String loanValuesJSON = gson.toJson(loanValuesMap);
//Convert the complex object values map JSON string into the actual object
loan = gson.fromJson(loanValuesJSON, com.salient.jbpm.dto.Loan.class);
System.out.println("ProcessCWithSignalStartJSON LoanNumber:::" + loan.getLoanNumber());
System.out.println("ProcessCWithSignalStartJSON callerProcessInstanceId:::" +
callerProcessInstanceId);
System.out.println("ProcessCWithSignalStartJSON signalCode:::" + signalCode);
kcontext.setVariable("loan", loan);
kcontext.setVariable("signalCode", signalCode);
kcontext.setVariable("loan", loan);
```

Issues:

If the child process needs to send a signal back to the parent process, pass the callback signal code along with the rest of the payload to the child process. The callback signal code must be unique to the caller. One way to uniquely identify the parent process is the parent process instance id.
processInstanceId = (Long) kcontext.getProcessInstance().getId();
If the signal code of the Catch Intermediate Signal Event in the parent is "childProcessComplete", the callback signal would be
callbackSignalCode = "childProcessComplete_" + processInstanceId;
This would enable a process to be started by multiple process instances and maintain runtime uniqueness.

2. Use the org.kie.api.runtime.process.ProcessRuntime API
kcontext.getKieRuntime().signalEvent(String signalCode, Object payload);
Payload:
The payload is constructed by the caller and sent as a parameter to the signalEvent API. The payload can be a java.util.Map to hold multiple variables.
Eg.

```
//Set up payload: loan, callbackSignalCode, and callerProcessInstanceId. Payload will be in the
//form of a Map //(params)
loan = new com.salient.jbpm.dto.Loan();
loan.setLoanNumber("1234");
com.salient.jbpm.dto.Person applicant = new com.salient.jbpm.dto.Person();
applicant.setFirstName("Thelonious");
applicant.setLastName("Monk");
loan.setApplicant(applicant);
loan.setLoanAmount(new Double(10000));
kcontext.setVariable("loan", loan);

//Get current process instance id
processInstanceId = (Long) kcontext.getProcessInstance().getId();
kcontext.setVariable("processInstanceId", processInstanceId);

//Signal code to start other process
signalCode = "startSignalProcess";
kcontext.setVariable("signalCode", signalCode);

//We could also send a unique callbackSignalCode ("signalProcessCompleted_" + processInstanceId)
//in which case the other process would use signalEvent(callbackSignalCode, payload), rather than
//signalEvent(callbackSignalCode, payload, processInstanceId)
callbackSignalCode =  "signalProcessCompleted";
kcontext.setVariable("callbackSignalCode", callbackSignalCode);

//Start process by signalEvent
java.util.Map params = new java.util.HashMap<String, Object>();
params.put("loan", loan);
params.put("callbackSignalCode", callbackSignalCode);
params.put("callerProcessInstanceId", processInstanceId);
org.kie.api.runtime.KieSession ksession = kcontext.getKieRuntime().getKieBase().newKieSession();
ksession.signalEvent("startSignalProcessC", params);
```

Issues:
If the process being started via the signalEvent API has a human task an exception is thrown: Could not find work item handler for Human Task.

If the process being started via the signalEvent API does not have a human task, no exception is thrown, but a process instance for the started process does not appear to be created. The process, however, does run to its conclusion.

3. Use the JMSSendTask work item handler
   Payload:
   Eg Payload set up for:

```
/*Send*/
//Send the calling processInstanceId so that the //receiving process can call back
processInstanceId = (Long)kcontext.getProcessInstance().getId();
callbackSignal = "processBWithSignalCompleted_" + processInstanceId;
kcontext.setVariable("callbackSignal", callbackSignal);

//Complex object in payload
loan = new com.salient.jbpm.dto.Loan();
loan.setLoanNumber("1234");
com.salient.jbpm.dto.Person applicant = new com.salient.jbpm.dto.Person();
applicant.setFirstName("Thelonious");
applicant.setLastName("Monk");
loan.setApplicant(applicant);
loan.setLoanAmount(new Double(10000));
kcontext.setVariable("loan", loan);

//Signal code to start the other process
signalCode = "startSignalProcess";
kcontext.setVariable("signalCode", signalCode);


//If signal receiver is expecting parameters to be passed via a Map
params = new java.util.HashMap<String, Object>();
params.put("loan", loan);
params.put("processInstanceId", processInstanceId);
params.put("callbackSignal", callbackSignal);
kcontext.setVariable("params", params);

/*Receive*/
//Get parameters via the params Map
loan = (com.salient.jbpm.dto.Loan)params.get("loan");
processInstanceId = (java.lang.Long)params.get("processInstanceId");
callbackSignal = (java.lang.String)params.get("callbackSignal");
kcontext.setVariable("loan", loan);
kcontext.setVariable("processInstanceId", processInstanceId);
kcontext.setVariable("callbackSignal", callbackSignal);
```

Conclusion

In conclusion, IBM BAMOE enables inter-process communication in a number of ways:

- Dependent inclusion by adding KIE bases
- REST using the built in KIE server API's
- JMS using messaging
- Action script using KIE Session API
- Directly nesting one service inside the definition of another service

The best method would be the one most appropriate to your project's design needs.

If you're interested in learning more, Contact Salient's Automation Advisors today!

[Contact Us | SalientProcess]

APPENDIX

I. Setting up REST calls

To set up a project to perform REST calls:
Create a client
1. In the process canvas
2. Select JBPM-WORKITEMS-REST from the palette
3. Select the REST step just added
4. Under Data Assignment in the right nav, add all necessary REST inputs and output mapping
5. In the Project view click on Settings
6. In the left nav click on Deployments
7. In the sub-menu click on Work Item Handlers
8. Click +Add Work Item Handler
9. Under Name enter "Rest", and under Value enter "new org.jbpm.process.workitem.rest.RESTWorkItemHandler()"

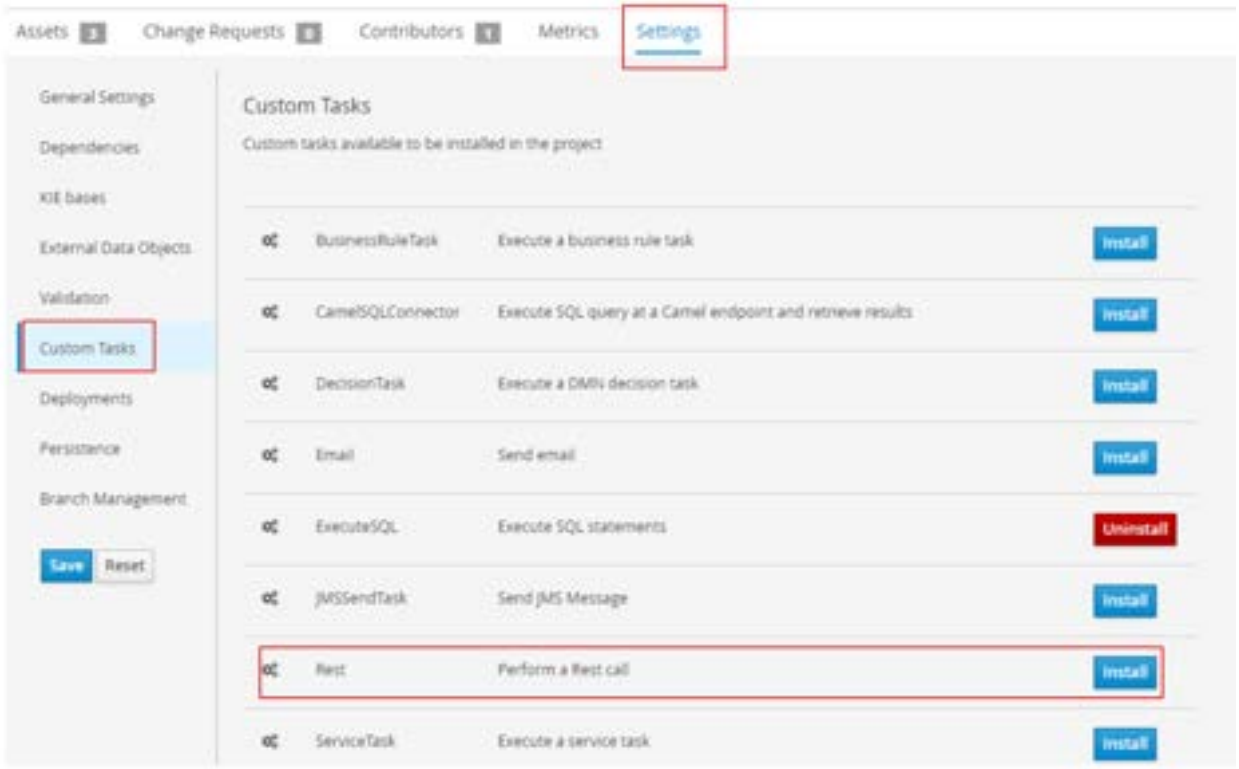
Exposing JBPM process as a REST service:
A JBPM process cannot be directly exposed as a REST service. However, there are KIE REST API's that can be used to start a process aysnchronously or synchronously.

/kie-server/services/rest/server/containers/{containerId}/processes/{processId}/instances
(Starts a new process instance of a specified process. The is an asynchronous API, the return value is the processInstanceId)

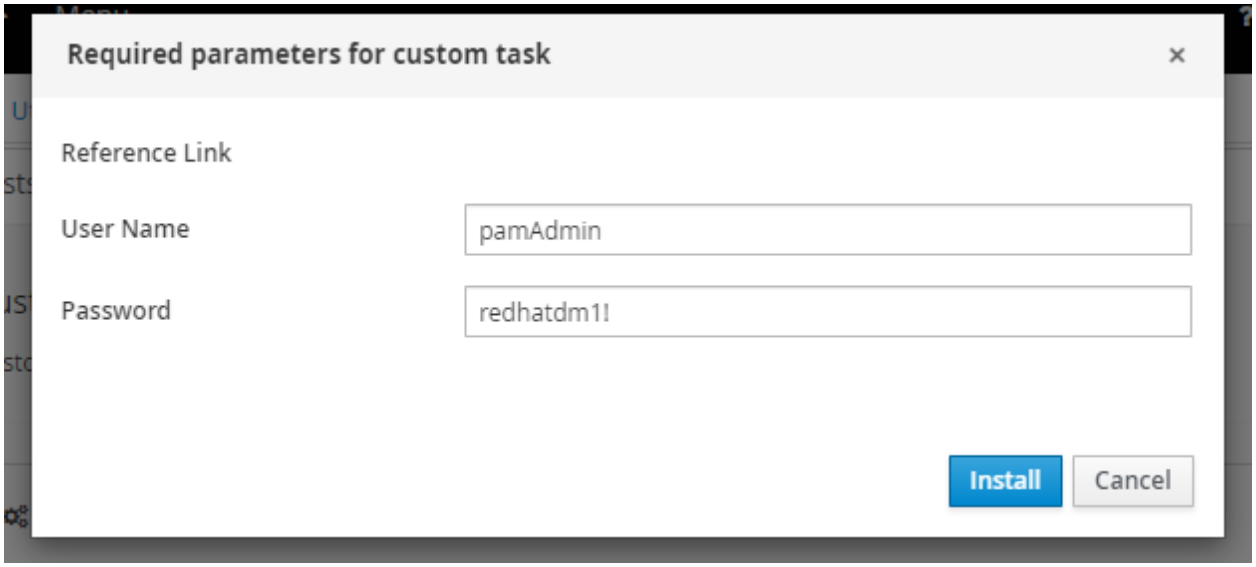
/kie-server/services/rest/server/containers/{containerId}/processes/{processId}/computedInstances
(Starts a new synchronous process instance of a specified process, returns the values of all the process variables.)

Steps with screenshots below:

1. Under project Settings/Custom Tasks Install the Rest workitem handler

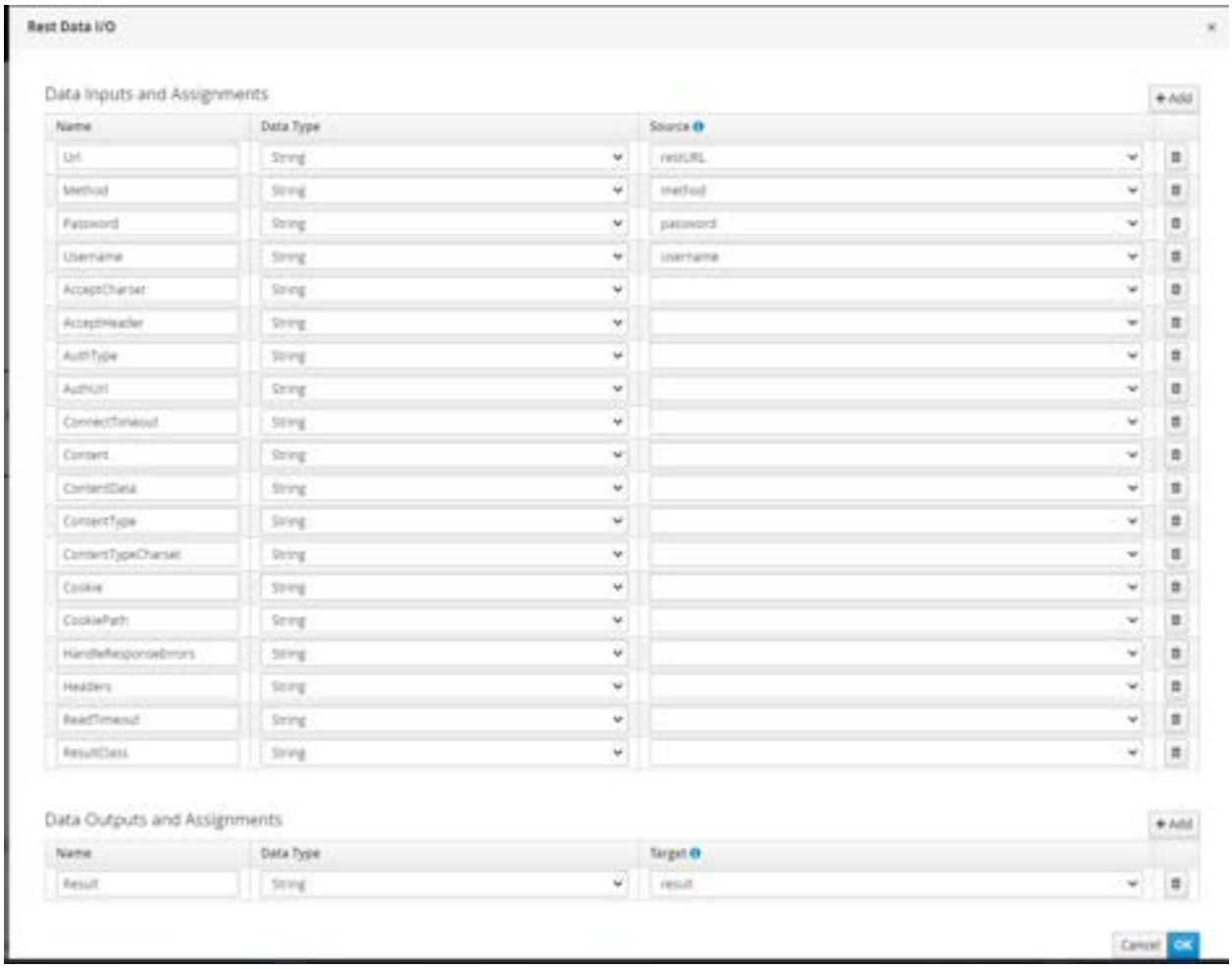


2. In the popup that follows enter the username and password for the Rest service. Click Install. In the Settings page click Save.



3. In a Business Process click on Custom Tasks in the palette. In the popup that follows select the Rest workitem.

Connect the REST workitem to the other components in the process, initialize variables as necessary and map the Rest inputs/outputs into the service. The inputs can be hardcoded or passed by a variable. Any variables passed into the Rest service must first be set in the kcontext variable. Eg. Kcontext.setVariable("restURL", restURL);



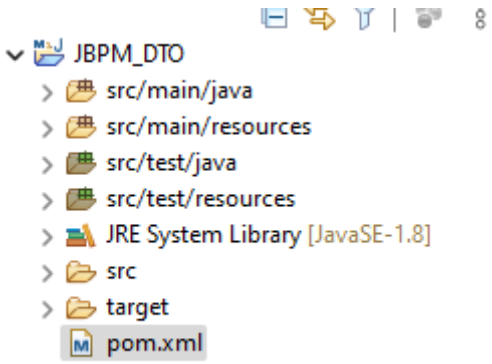
—----------Setting up REST calls----------

II. <u>Data Objects created outside BAM</u>

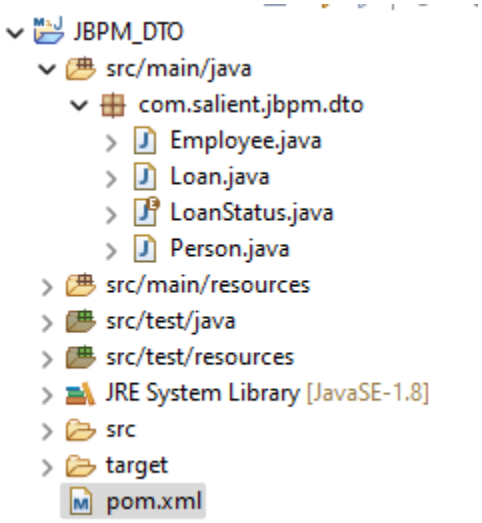
Data Objects created in a project are available within that project. An alternative to creating Data Objects in Business Central is to create the objects (Java classes) outside of Business Central and import the objects into Business Central and then add them as dependencies at the project level.

To create data objects outside of Business Central using Eclipse:

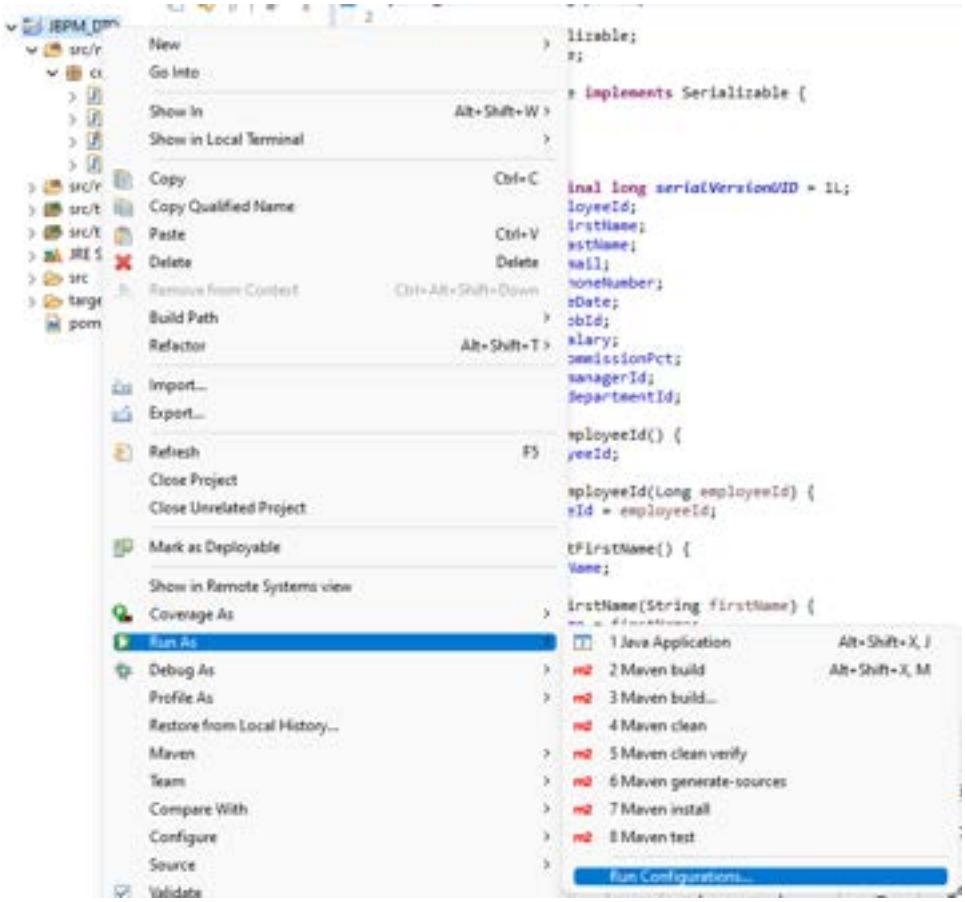
1. Create a Maven project



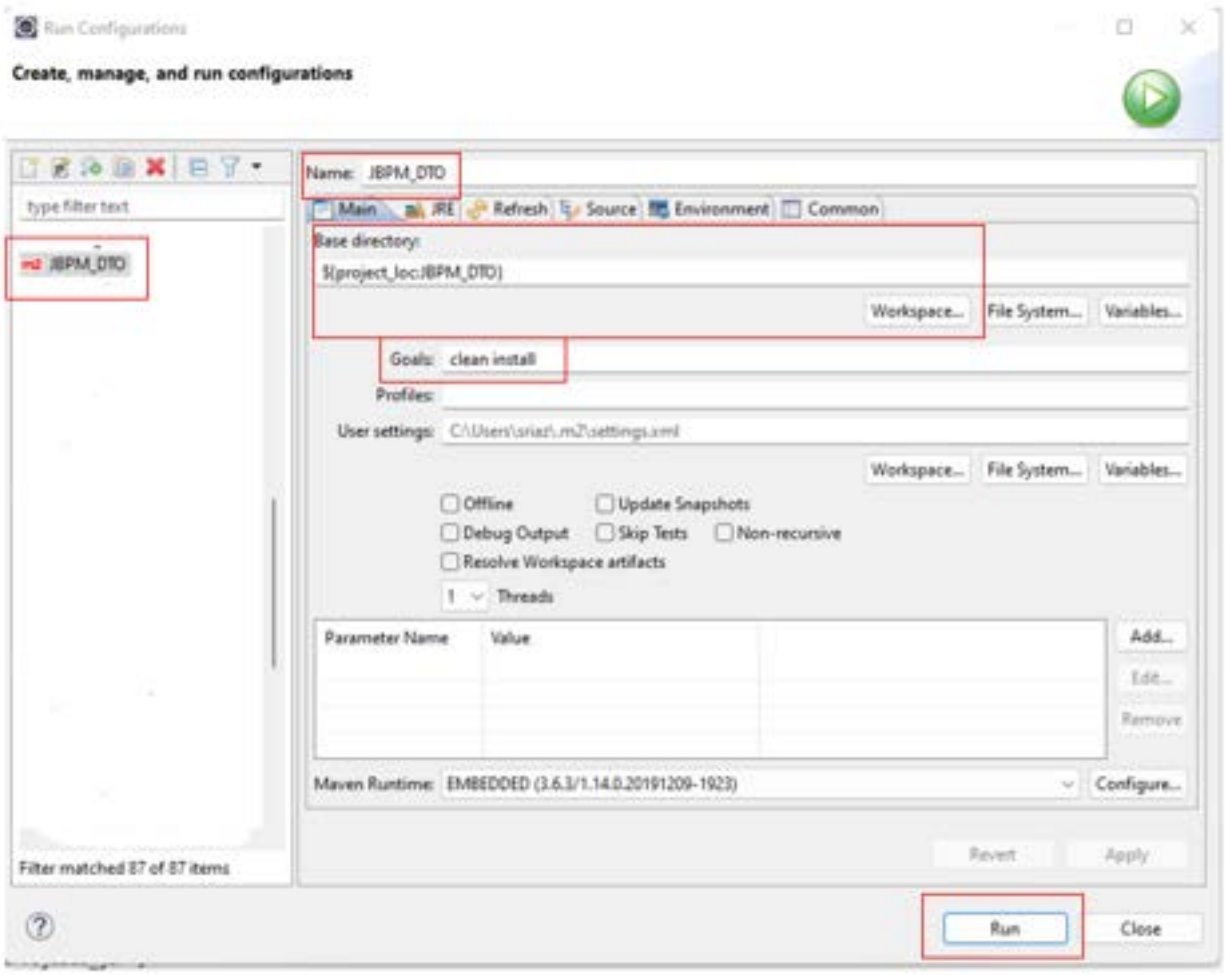
2. Create classes under this project



3. The structure of these classes is similar to those generated when a Data Object is created in Business Central.
4. Build the project in Eclipse: right click on the project in the Project Explorer, click Run As, click Run Configurations...

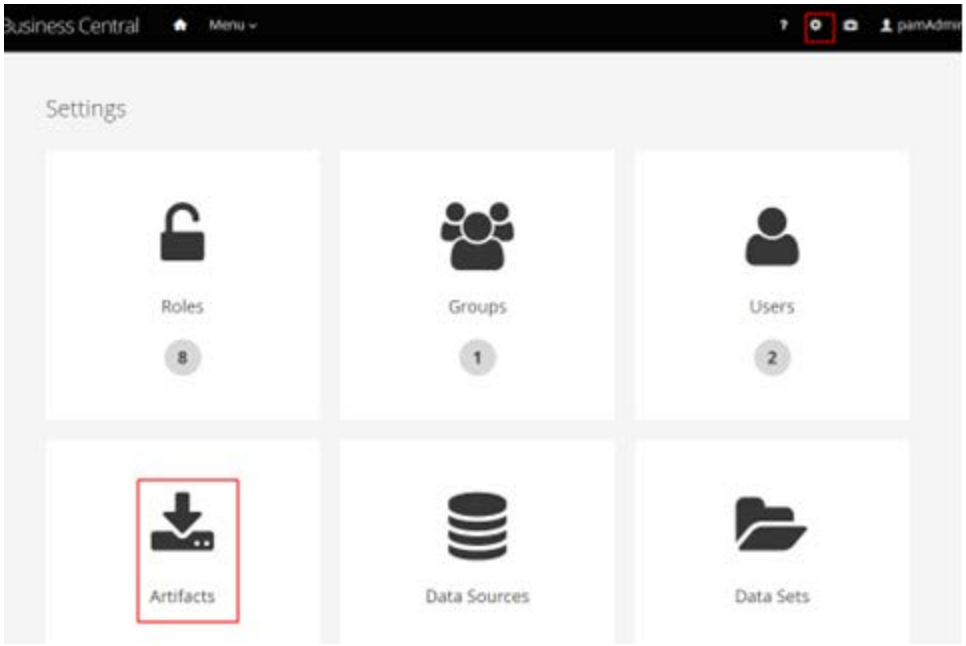


In the Run Configurations... window: under Maven Build in the left nav, give the new configuration a name; under Base directory click on Workspace and navigate to and select the project folder; under Goals enter “clean install”; click Run.

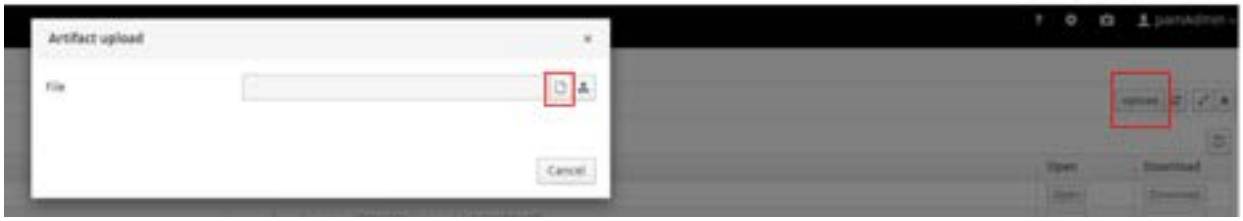


A successful build will produce a console output similar to the one below:

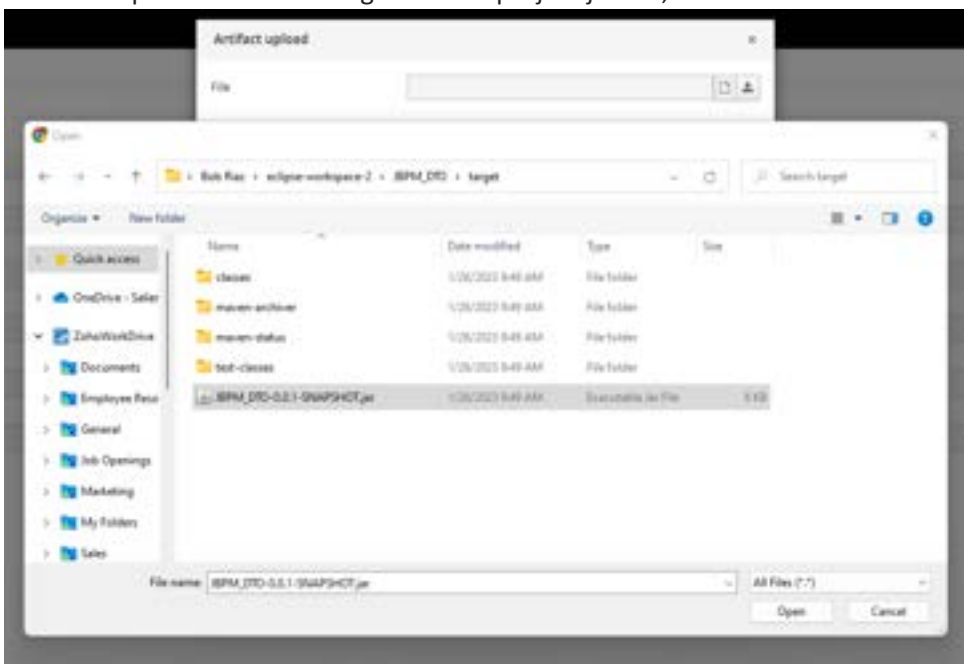
5. Upload artifact to Business Central and add the dependency to your project
   In Business Central click Settings (the gear icon in the top right corner), then click Artifacts



6. In the Artifacts page click Upload; in the Artifact upload popup click the Choose file... icon

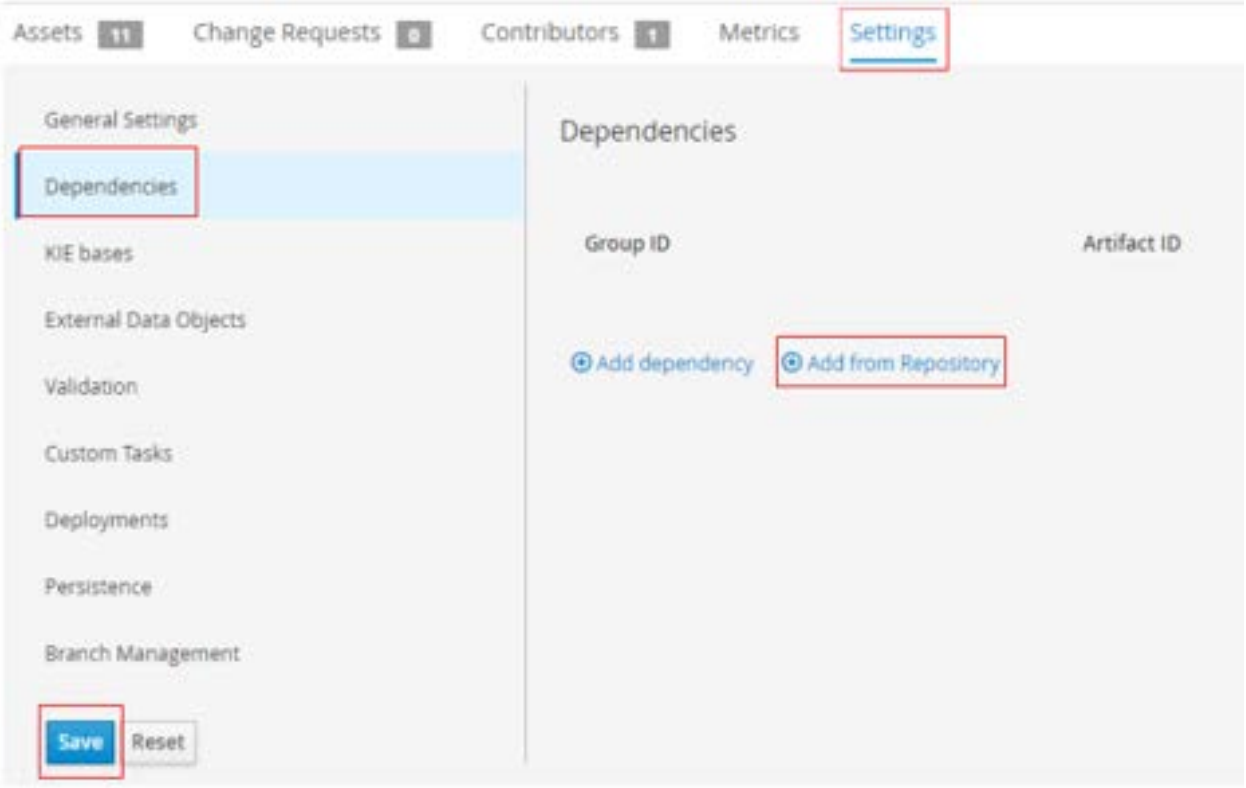


7. In the explorer window navigate to the project jar file, double-click the file or click Open.



8. In the Artifact upload window click the upload icon

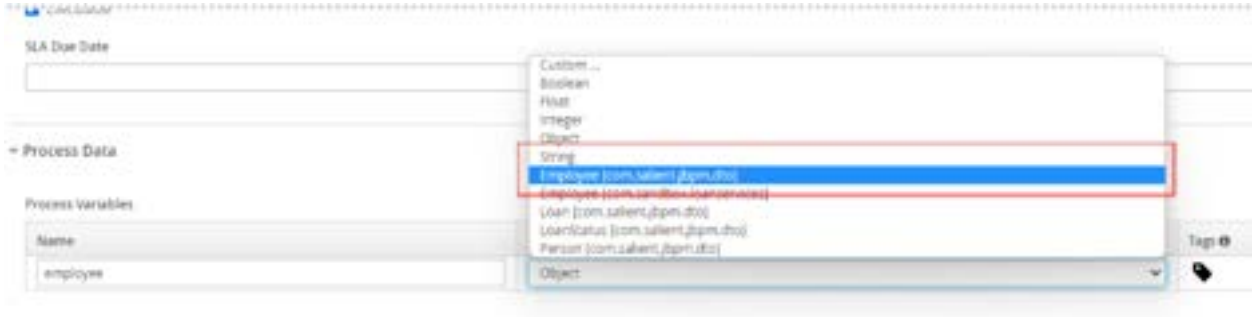
9. In your project, click the Settings tab; click Dependencies; click Add from repository



10. The jar uploaded in the previous step should be available to be added as a dependency. Search for the artifact by name and Select the latest version.

11. The jar will appear in the dependency list. Save.

12. The classes defined in the jar are now available to be used in the project. In any Business Process in the project, create a Process Variable. You will be able to select the class defined in the dependency as the type of the process variable. The variable can be used just like a Data Object defined in the project.



—----------Data Objects created outside BAM—----------

Set up JMS

Documentation:

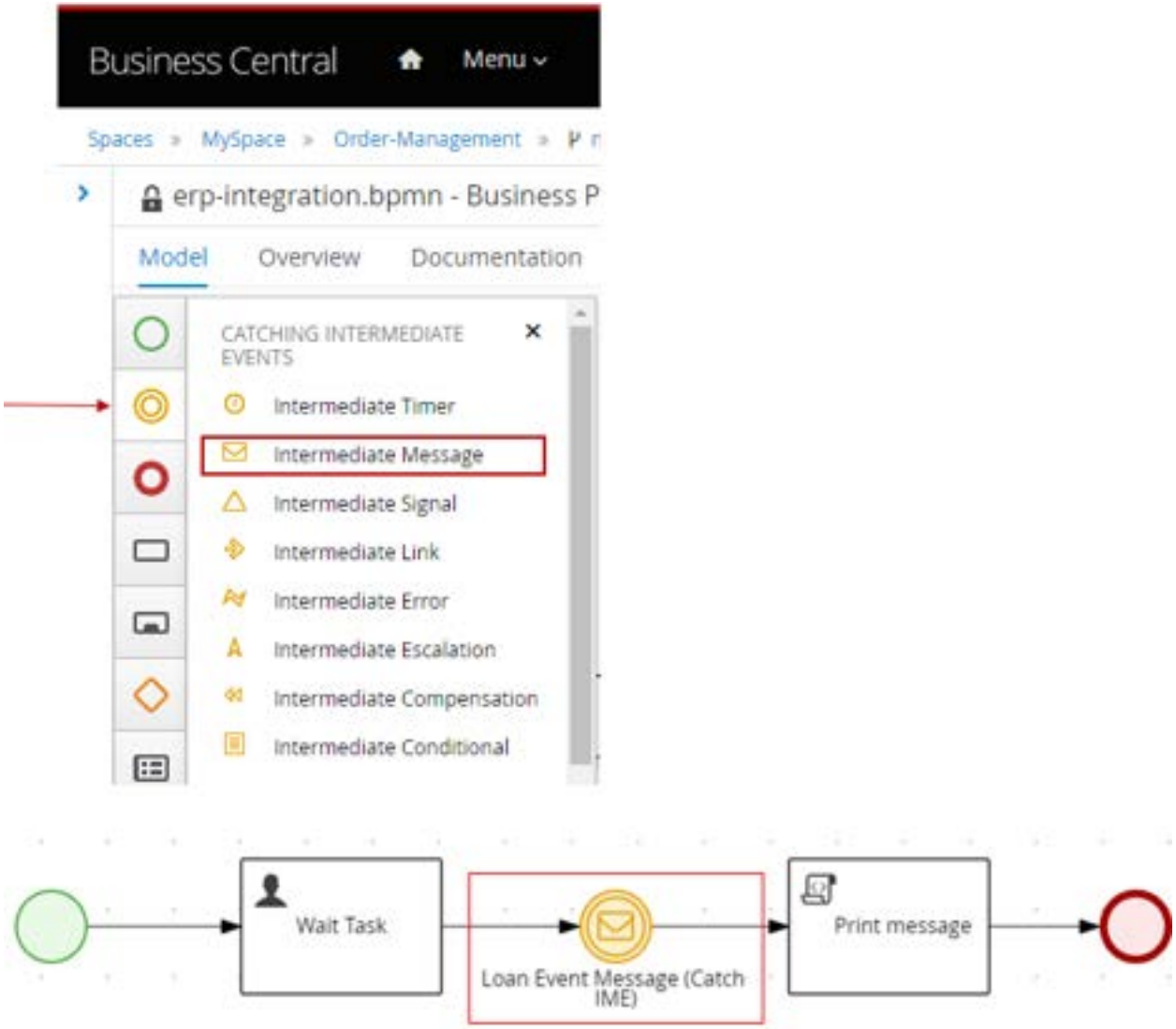
Message Intermediate Event (JBoss)

Message Intermediate Event (RedHat)

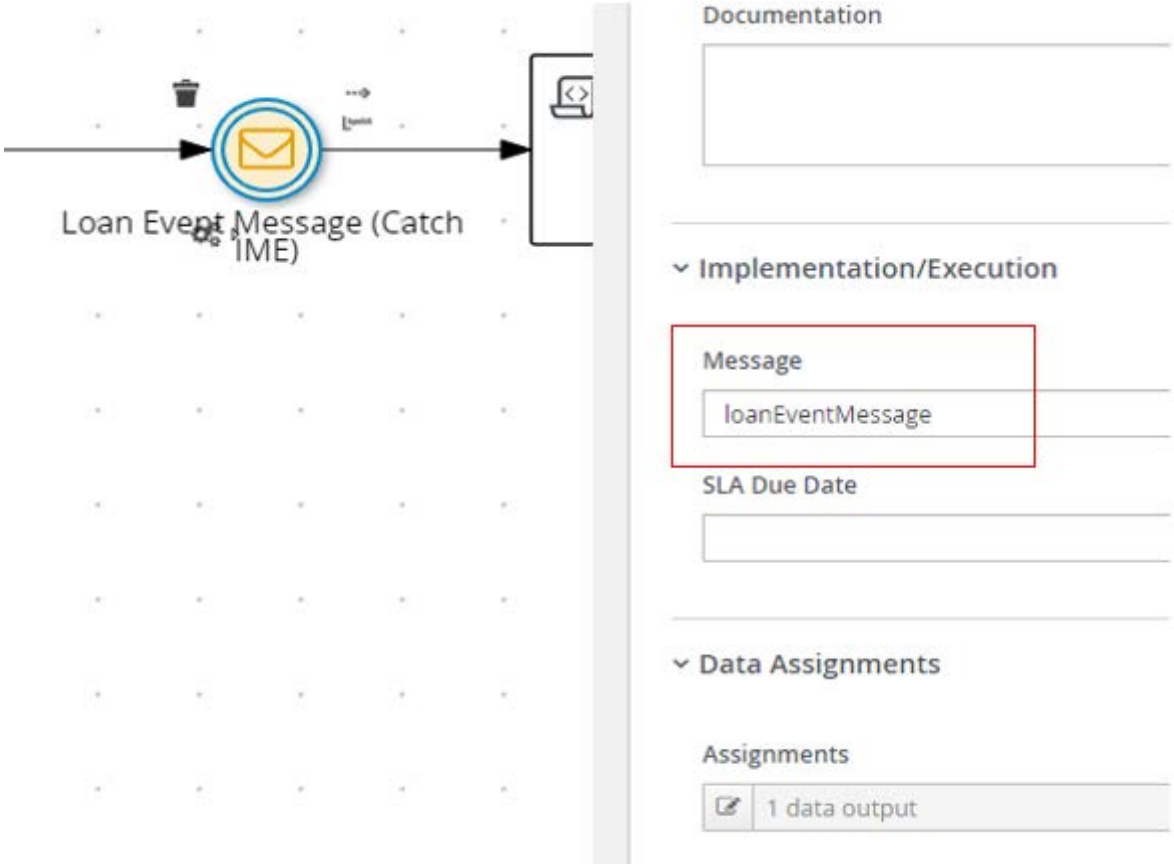
Below are the steps taken to set up a process with a Catch IME and a process with a JMSSendTask to send the message. Also included are the snippets of configuration files to support messaging and durable

subscriptions. Note: Non-durable subscription messaging works – when the message is sent after the Catch IME is active with a token. Durable subscription does not work – when the message is sent before the Catch IME is active with a token.

1. Process with Catch IME (Loan Event Message Process):



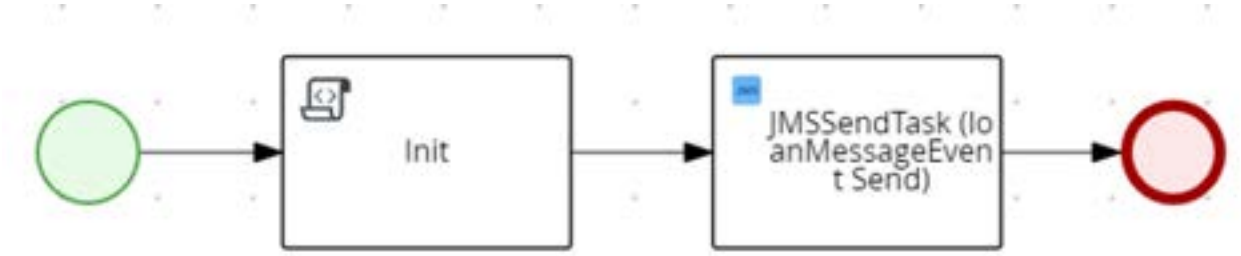
2. Catch IME set up:



3. Catch IME Data I/O: The messagePayload is the data sent by the message sender.

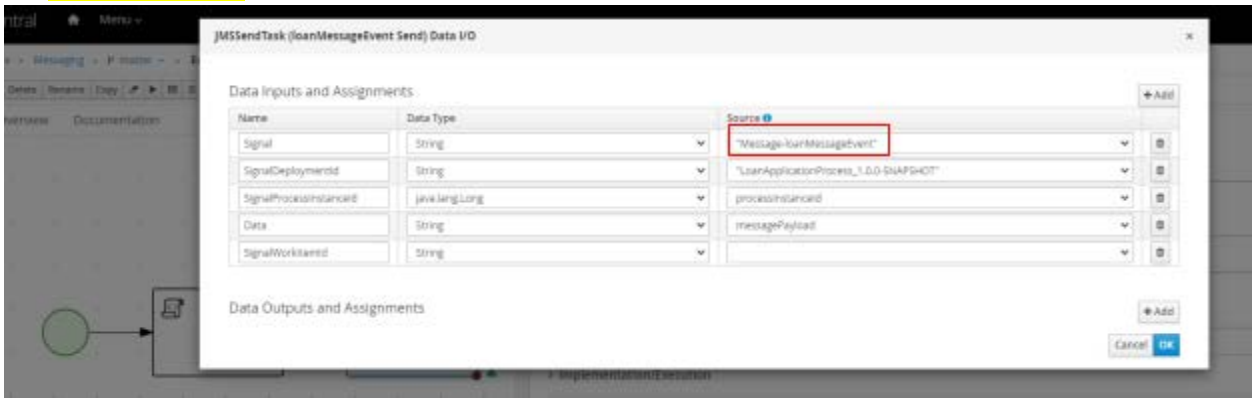
Send Message via JMSSendTask work item handler because Throw IME does not work.

1. Process set up to send message via JMSSendTask (External Message Send Process).

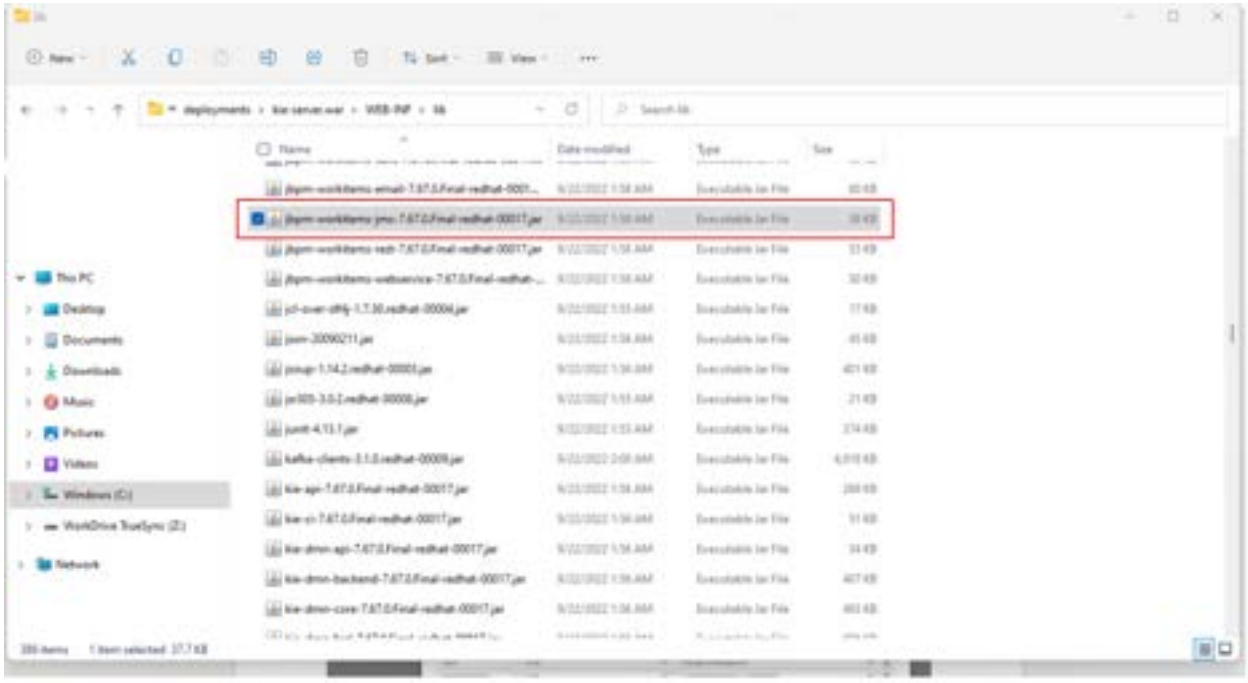


2. Set up inputs to JMSSendTask:

3. JMSSendTask data mappings. Note: When a message is sent the message value must start with "Message-"



4. In order to use the JMSSendTask work item handler the jbpm-workitems-jms jar must be available in the installation. RHPAM installs with this jar but the jar is also available in the Maven repository: https://mvnrepository.com/artifact/org.jbpm/jbpm-workitems-jms
5. Confirm the jbpm-workitems-jms jar exists in …\standalone\deployments\kie-server.war\WEB-INF\lib



6. Locate …\standalone\deployments\kie-server.war\WEB-INF\ejb-jar.xml, and uncomment the <message-driven> element.

```
<!-- enable when external signals are required and queue and connection factory is defined -->
  <message-driven>
    <ejb-name>JMSSignalReceiver</ejb-name>
    <ejb-class>org.jbpm.process.workitem.jms.JMSSignalReceiver</ejb-class>
    <transaction-type>Bean</transaction-type>
    <activation-config>
      <activation-config-property>
        <activation-config-property-name>destinationType</activation-config-property-name>
        <activation-config-property-value>javax.jms.Queue</activation-config-property-value>
      </activation-config-property>
      <activation-config-property>
        <activation-config-property-name>destination</activation-config-property-name>
        <activation-config-property-value>java:/queue/KIE.SERVER.SIGNAL</activation-config-property-value>
      </activation-config-property>
    </activation-config>
  </message-driven>
```

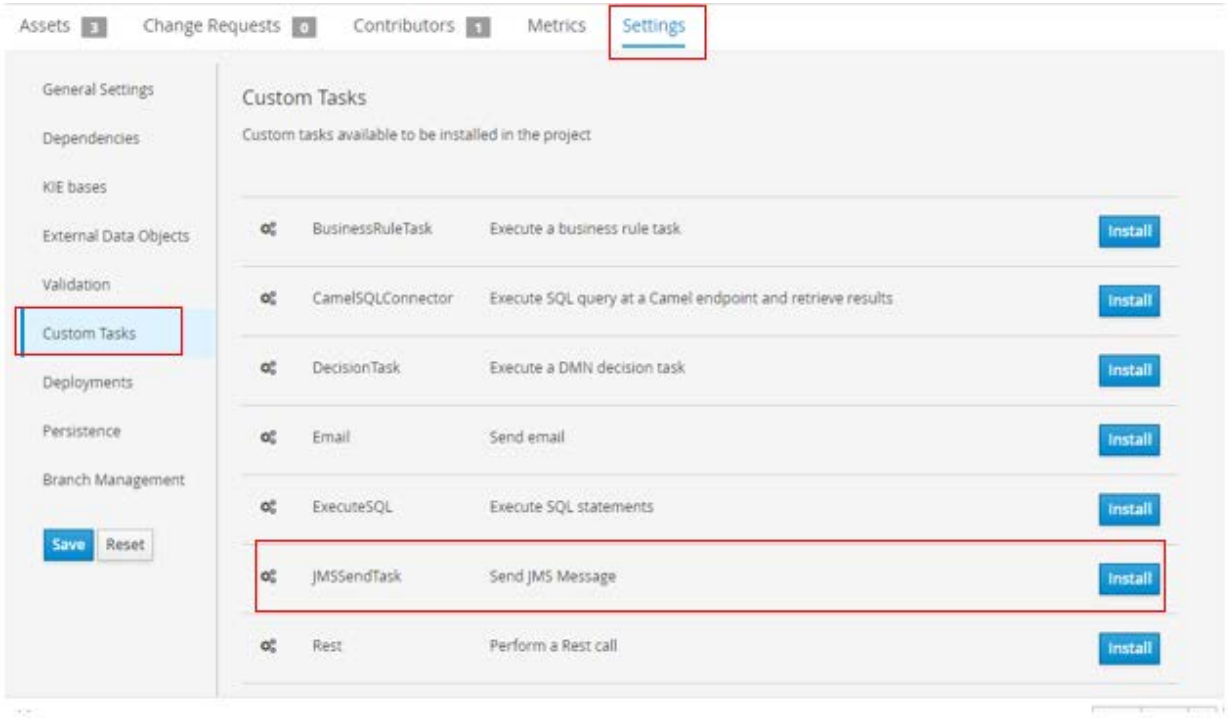
7. In …\standalone\configuration\standalone-full.xml locate the messaging-activemq subsystem:

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:13.0">
        <server name="default">
```

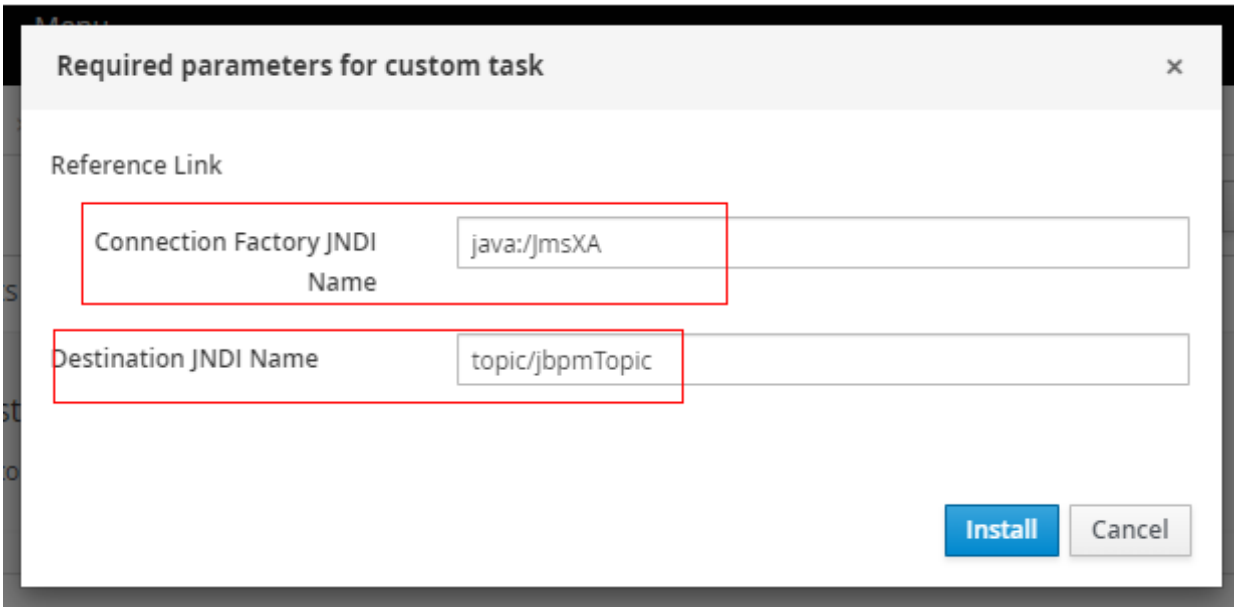
8. Locate the queues, topics and connections section of the subsystem. You can add queues and topics here. To enable communication between processes defined in different projects add the queue from the ejb-jar.xml above. Also add a topic element. The "entries" attribute holds the JNDI names of the element.

```
<jms-queue name="ExpiryQueue" entries="java:/jms/queue/ExpiryQueue"/>
<jms-queue name="DLQ" entries="java:/jms/queue/DLQ"/>
<jms-queue name="KIE.SERVER.SIGNAL.QUEUE" entries="java:/queue/KIE.SERVER.SIGNAL"/>
<jms-topic name="jbpmTopic" entries="topic/jbpmTopic jms/topic/jbpmTopic java:jboss/exported/jms/topic/jbpmTopic"/>
<connection-factory name="InVmConnectionFactory" entries="java:/ConnectionFactory" connectors="in-vm"/>
<connection-factory name="RemoteConnectionFactory" entries="java:jboss/exported/jms/RemoteConnectionFactory" connectors="http
<pooled-connection-factory name="activemq-ra" entries="java:/JmsXA java:jboss/DefaultJMSConnectionFactory" connectors="in-vm"
```

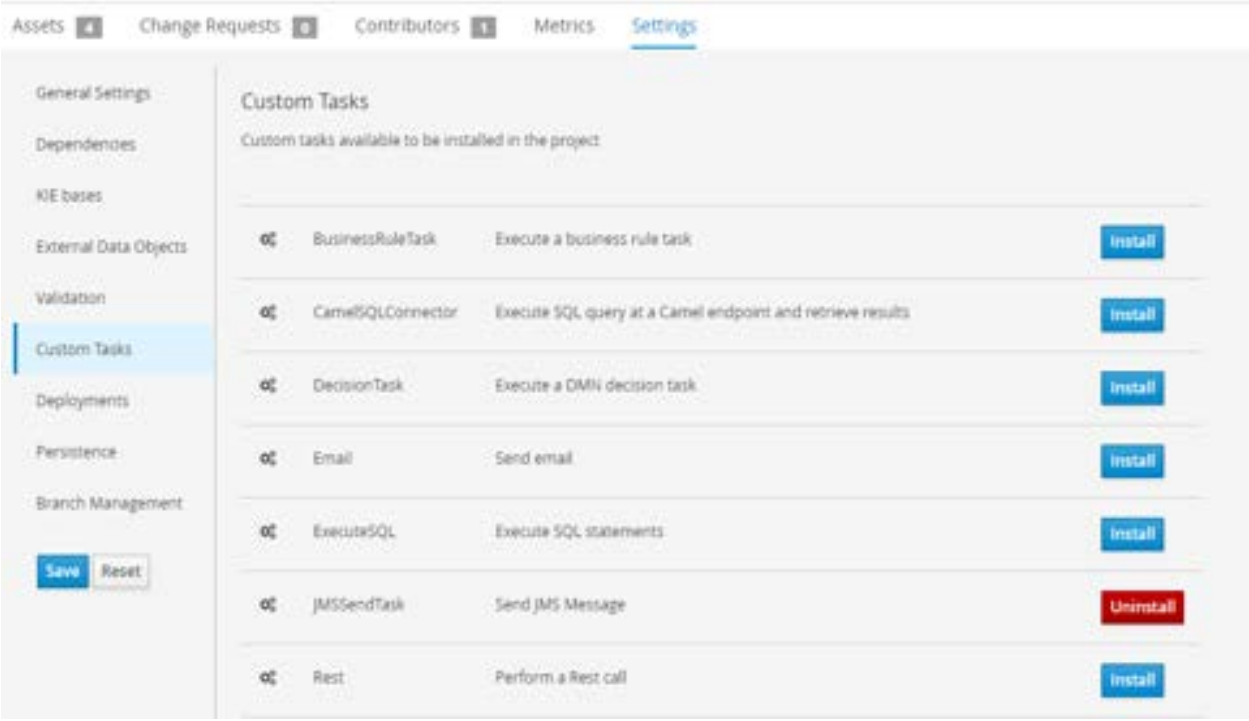
9. Install JMSSendTask work item handler in project Settings:



10. Click Install. In the popup window, enter the Connection Factory JNDI and Destination (topic or queue) JNDI) from the standalone-full.xml. Click Install. Note: Per the ActiveMQ and JMS specifications, Topics support durable subscriptions. However, this feature is currently not working in RH PAM. IBM has been notified of the issue.

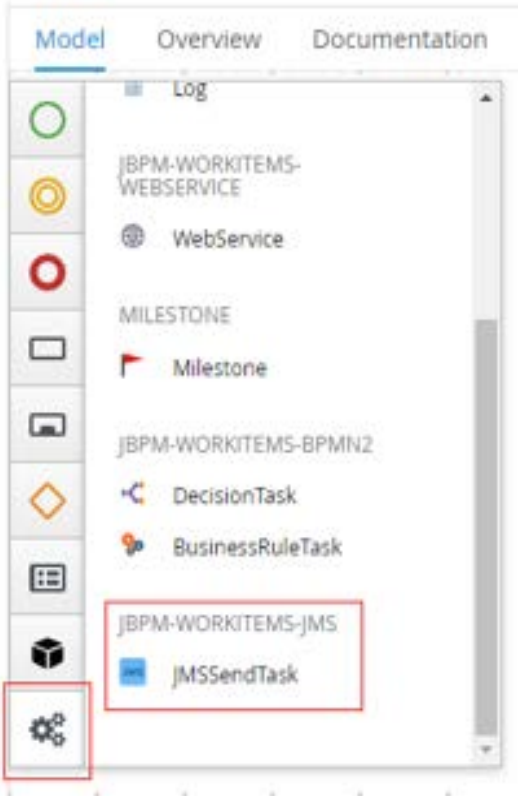


11. In the project Settings page, click Save:



12. Create a Business Process in the project.

13. In the process editor click Custom Tasks, select the JMSSendTask component and add it to the canvas.

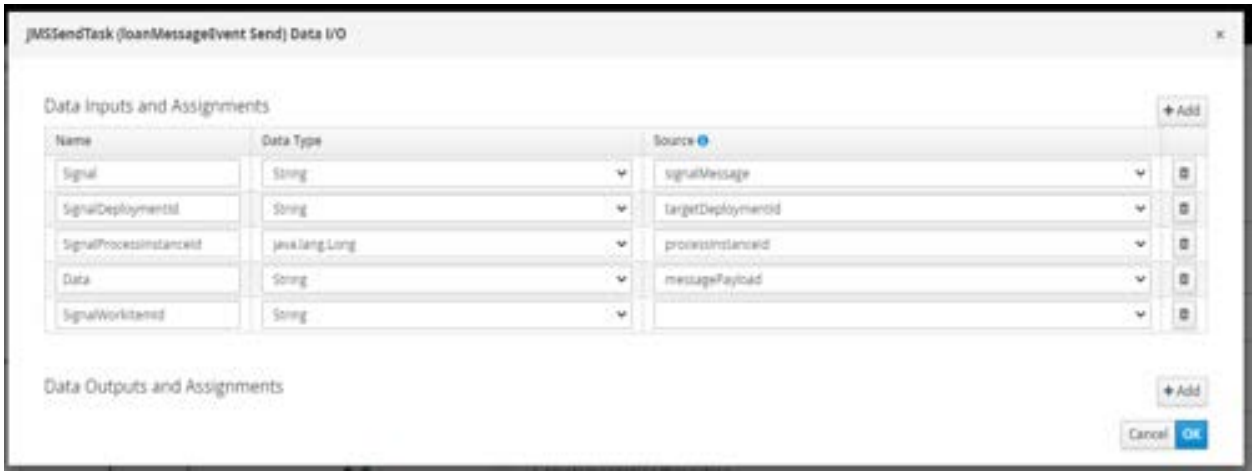


14. Connect the JMSSendTask component to the other components in the process

15. Create process variables that will be used in the JMSSendTask work item handler.

16. Initialize the variables

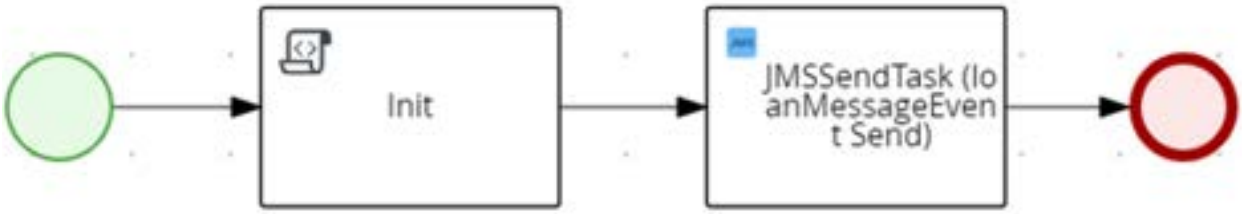
17. Map the data variables into the JMSSendTask work item handler.



Test Case Scenario:

Process with Catch IME (Loan Event Message Process):

Process set up to send message via JMSSendTask (External Message Send Process).



Non-Durable Subscription test:

1. Loan Event Message Process is started. Process instance is at Wait Task. Process instance id is generated
2. Wait Task of Loan Event Message Process is completed. Token moves to Catch IME.
3. External Message Send Process is started. Process instance id of Loan Event Message Process instance is entered in the form and submitted.
4. Loan Event Message Process IME correlates on the message and the process moves forward

Durable Subscription test (not working yet, question out to IBM):

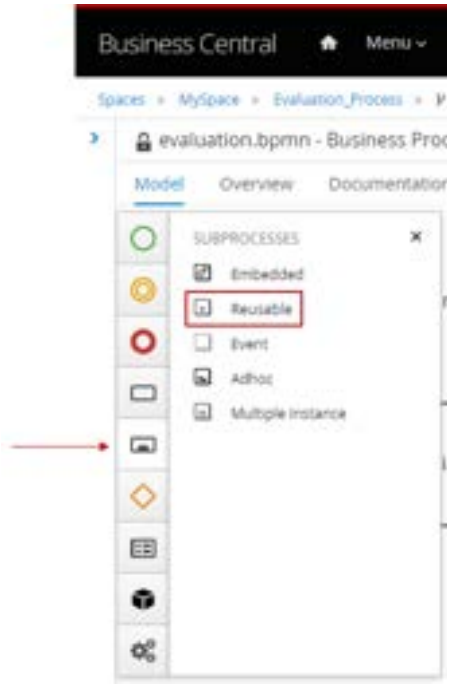
1. Loan Event Message Process is started. Process instance is at Wait Task. Process instance id is generated
2. External Message Send Process is started. Process instance id of Loan Event Message Process instance is entered in the form and submitted.
3. Wait Task of Loan Event Message Process is completed. Token moves to Catch IME.
4. Expected result: Catch IME in Loan Event Message Process should pick up the message that was sent in step 2 and move the process forward.
5. Actual result: The token does not move from the Catch IME in Loan Event Message Process.

Note: Signals can be used in place of Messages. The only difference is that when sending a signal the name of the signal does not start with "Message-".

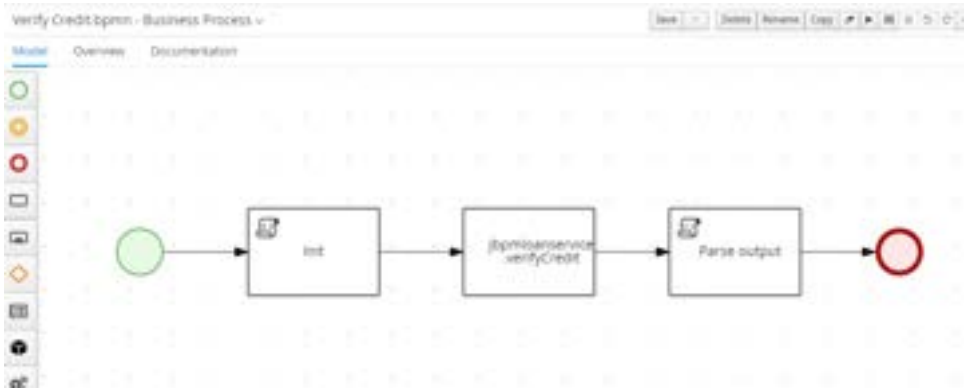
—----------Set up JMS—----------

Reusable

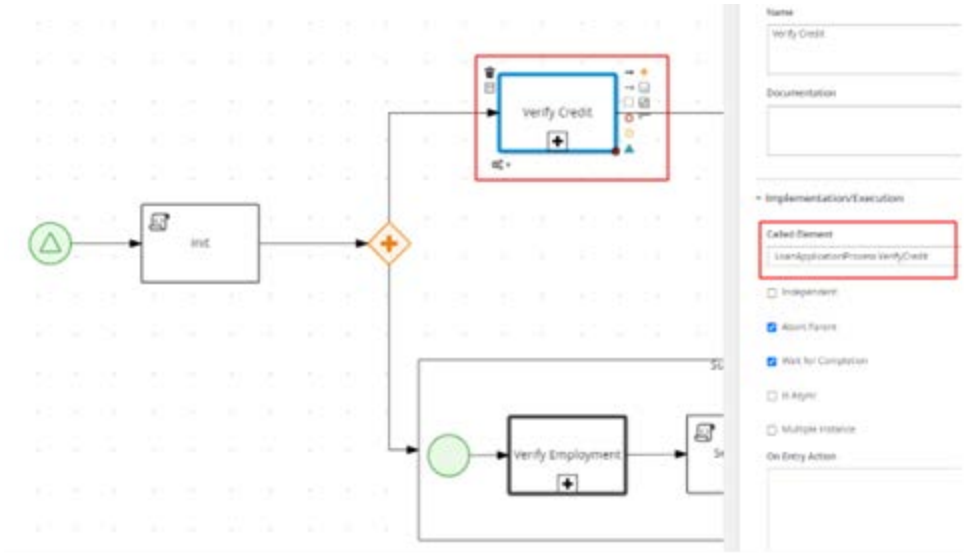
A business process that is independently defined within a project may be used in another process in the same project.



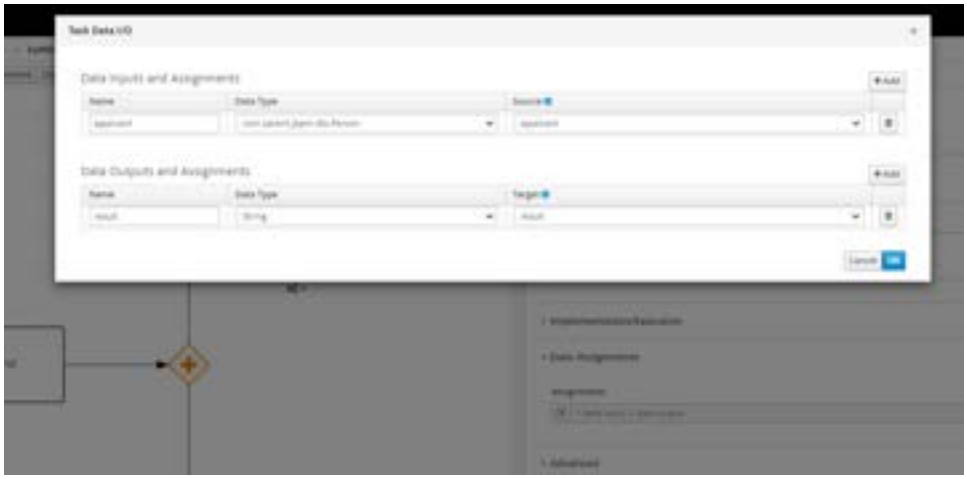
Eg.
Independent definition of a process called **Verify Credit**



Verify Credit used as a Reusable service in a different process. In the configuration options for the Reusable service step, select the process you wish to use in the Called Element drop-down.



Data can be mapped into and out of the Reusable service.



Direct reuse as in the example above seems to be possible only if both processes are defined in the same project. The Called Element drop-down will include processes defined in other projects, but selecting one of these fails at runtime. To invoke a process defined in one project from a process defined in another project use a REST integration with the appropriate KIE REST API:
/server/containers/{containerId}/processes/{processId}/instances
/server/containers/ {containerId} /processes/instances/signal/{signalName}
/server/containers/{containerId}/processes/instances/{processInstanceId}/signal/{signalName}
/server/containers/{containerId}/processes/{processId}/computedInstances
—----------Reusable—----------