Building Your First Process with Oracle BPM 11g

This tutorial contains the following sections:

Purpose

This tutorial shows you how to build a simple Hello World application using Oracle BPM Suite 11gR1. It also shows you how to deploy the process to the BPM engine and test it in the BPM Workspace.

Time to Complete

Approximately 2 hours

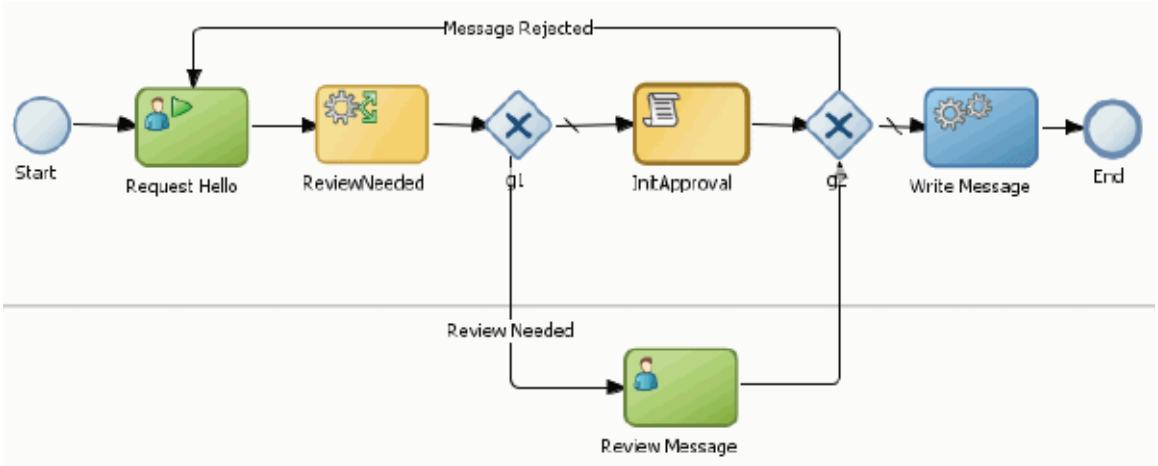
Overview

In this tutorial, you use Studio, the JDeveloper based IDE, to create a simple Hello World process. This process demonstrates the use of a file service, interactive tasks implemented by the human workflow engine, and by conditional branching. The conditions for the conditional branching are determined through the use of a business object and a business rule. You also use a script task to initialize a variable. After building the process, you deploy it to the BPM engine and test it in the runtime environment.

Scenario

There are two roles involved in the Hello World process, the sender of the message, acting in the Requester role, and a reviewer, acting in the Reviewer role. The requester is prompted, through the Request Hello activity, to enter a Hello message, greeting, and a date for the message. After the form is submitted, a business rule is applied to the message content to determine whether the message requires a review, based on the length of the greeting and message fields.

If the message does not require review, the process flows to a script task, which initializes a variable needed by the next task, then the message is sent to the Write Message activity to be written to the file system. If the message requires review, the reviewer is prompted to review the message and either accept or reject it. If the message is rejected, it returns to the Request Hello activity so that the requester can correct the message, otherwise, it goes directly to the Write Message activity for file processing.



Software and Hardware Requirements

In order to perform this tutorial, you must have previously installed Oracle BPM 11gR1 and JDeveloper 11.1.1.3 with both the SOA and BPM extensions. You will also need to have at least one user in the internal LDAP database of your WebLogic server in the OBPM installation in order to map this user to the roles that you define in your Hello World process. You can take care of both of these tasks (installation and seeding of the LDAP database) by performing the **Installing Oracle BPM 11g** OBE.

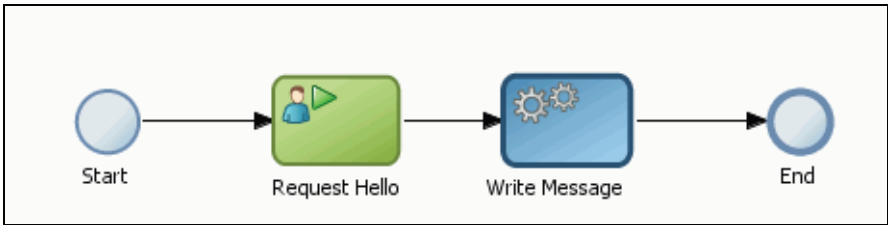
Prerequisites

If you have not yet installed OBPM 11gR1, perform the **Installing OBPM 11g** OBE. Performing this OBE will also seed your LDAP database.

If you already have an OBPM 11gR1 installation, but still wish to seed the Demo Community in the WebLogic server's internal LDAP database, download the zip file containing ANT files needed to perform this task. This is available from OTN as a **SOA 11g Human Workflow** sample code download. You will need to modify some parameters in the ANT build file to match your particular installation. A ReadMe file is included in the zip file to assist you.

Creating the Basic Hello World Application

In this section you create the basic starting point for the Hello World process using the JDeveloper Studio. You add two activities - an interactive activity and a service activity. The end user will be able to enter a Hello message, using the BPM Workspace. The message will be captured in a business object and passed to a file service, which, in turn will write the message to a disk file. Later, you expand upon this to add more complexity to the process.



You create several process elements throughout this section of the tutorial. The following naming convention will be used throughout this section:

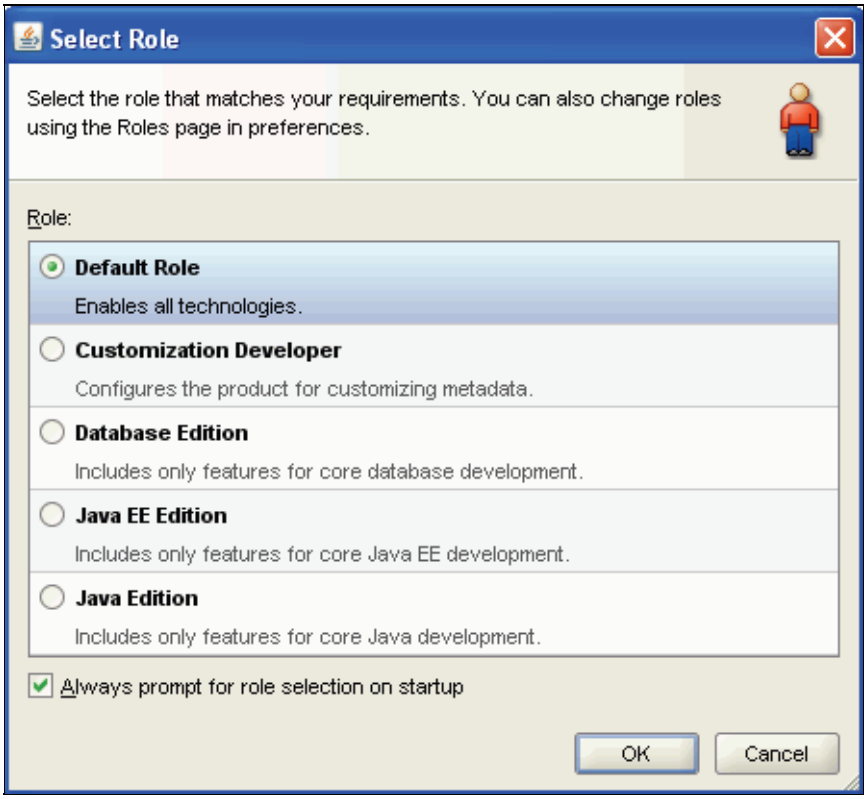
Name	Description
`HelloWorld_OBE`	Application name
`HelloWorldProject`	Main project name
`HelloWorld_UI`	Project containing user task web form(s)
`HelloWorldProcess`	Process name

Creating the Process Model
Creating the Business Object

Implementing the User Task
Implementing the File Service

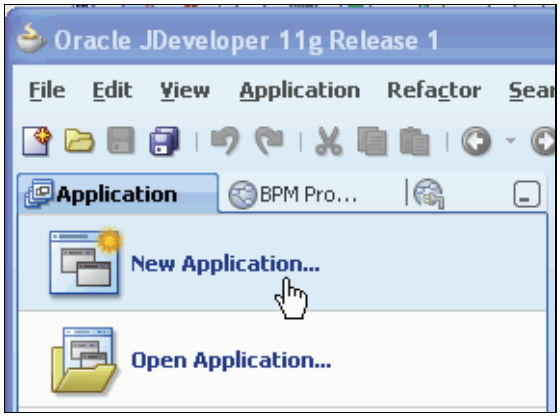
Creating the Process Model

1. Open **JDeveloper Studio 11.1.1.3** from the Windows Start menu. When prompted to select a role, choose the **Default Role**. Click **OK**.

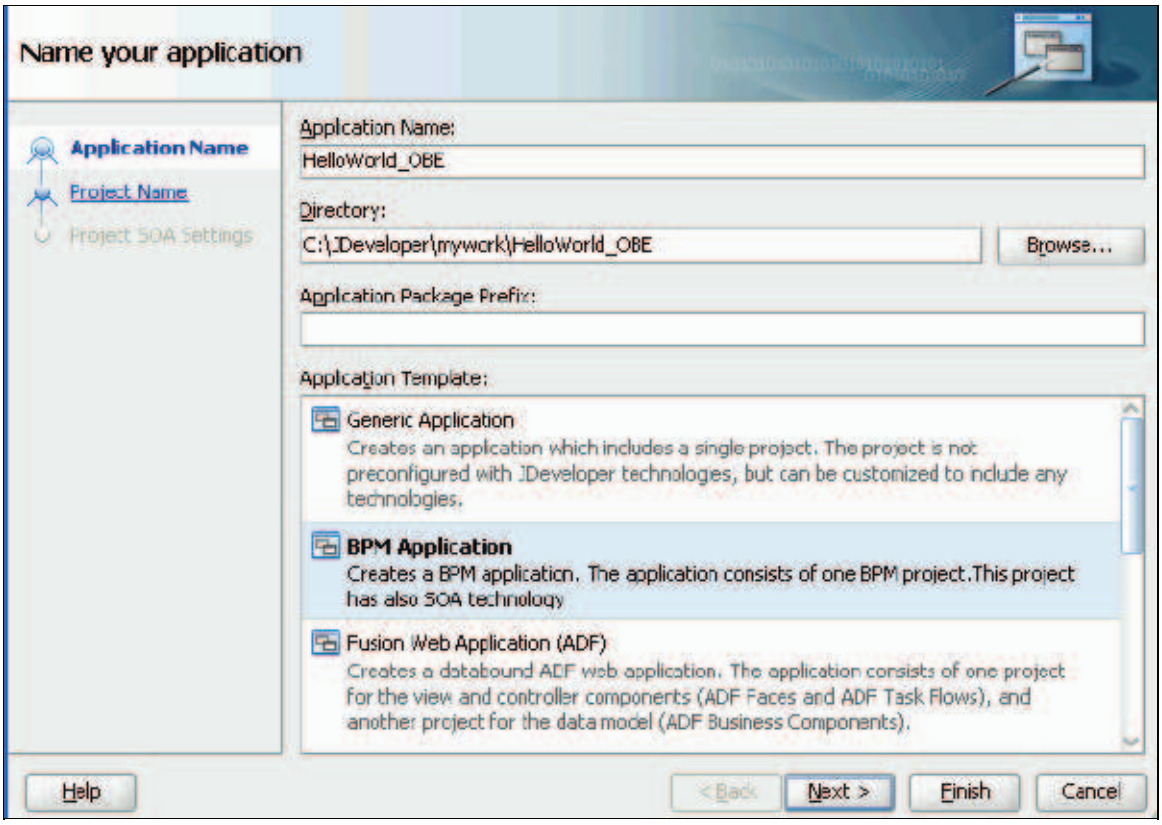


Close the Daily Tips window.

2. Create a new application. Click the **New Application** bar in the left panel.

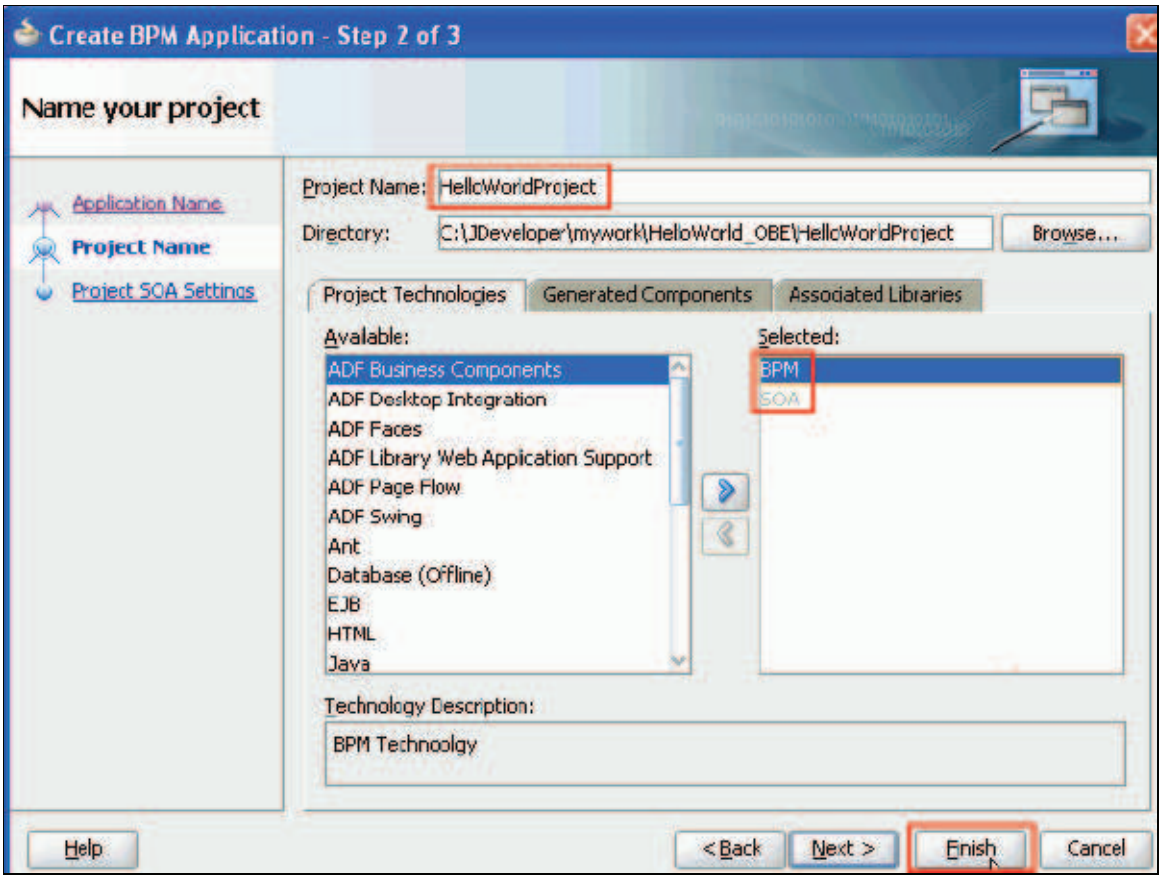


The BPM Application wizard opens. Name the application "**`HelloWorld_OBE`**" and accept the default directory for storing application files (`C:\JDeveloper\mywork`). Select **BPM Application** in the Application Template panel.

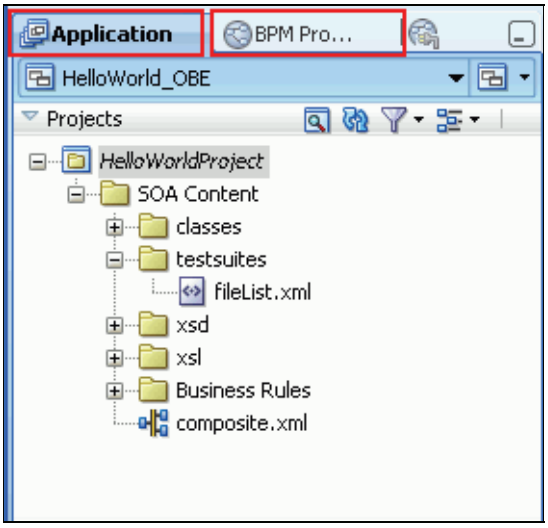


Click **Next**.

3. In **Step 2** of the **Create BPM Application** wizard, you create a project for the HelloWorld_OBE application. Enter `HelloWorldProject` as the Project Name. Notice that **BPM** and **SOA** are selected as project technologies by default. Click **Finish**.

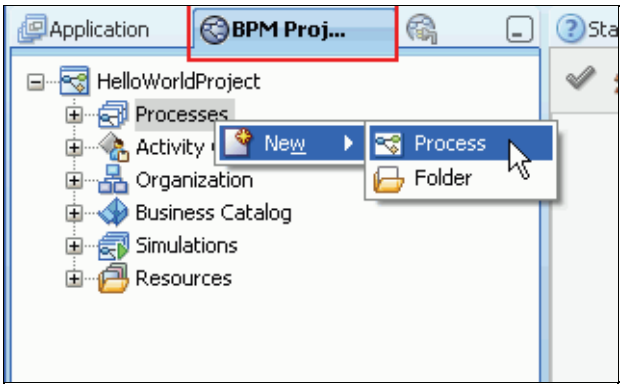


In the upper left corner of the JDeveloper Studio window, you see the Navigator panel. This contains two tabs that will be important to you as you perform this tutorial: The **Application Navigator** tab and the **BPM Project Navigator** tab. Currently the Application Navigator tab is selected by default. You can see the HelloWorld_OBE application appearing in the drop-down list just above the panel and the HelloWorldProject appearing as the parent node within the panel. The fact that it appears in *italics* indicates that there are unsaved changes.

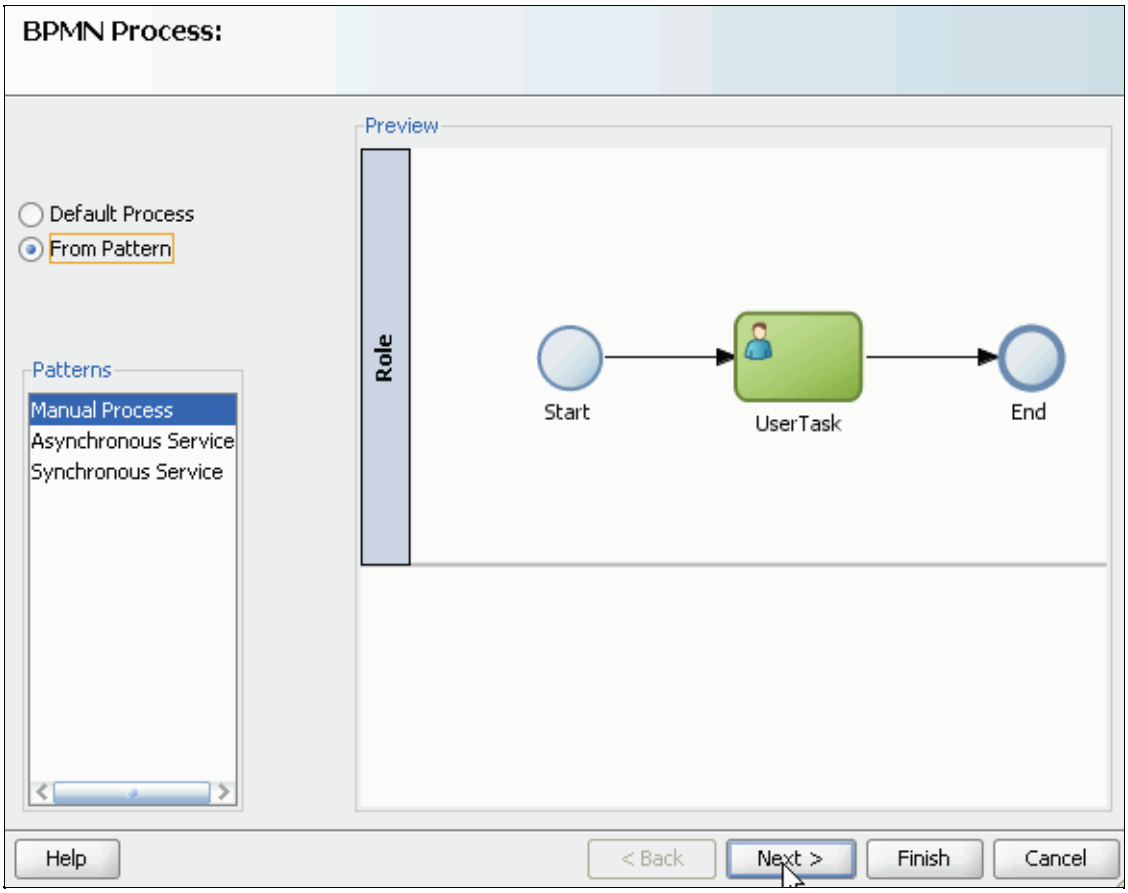


Click the **Save All** icon on the main toolbar.

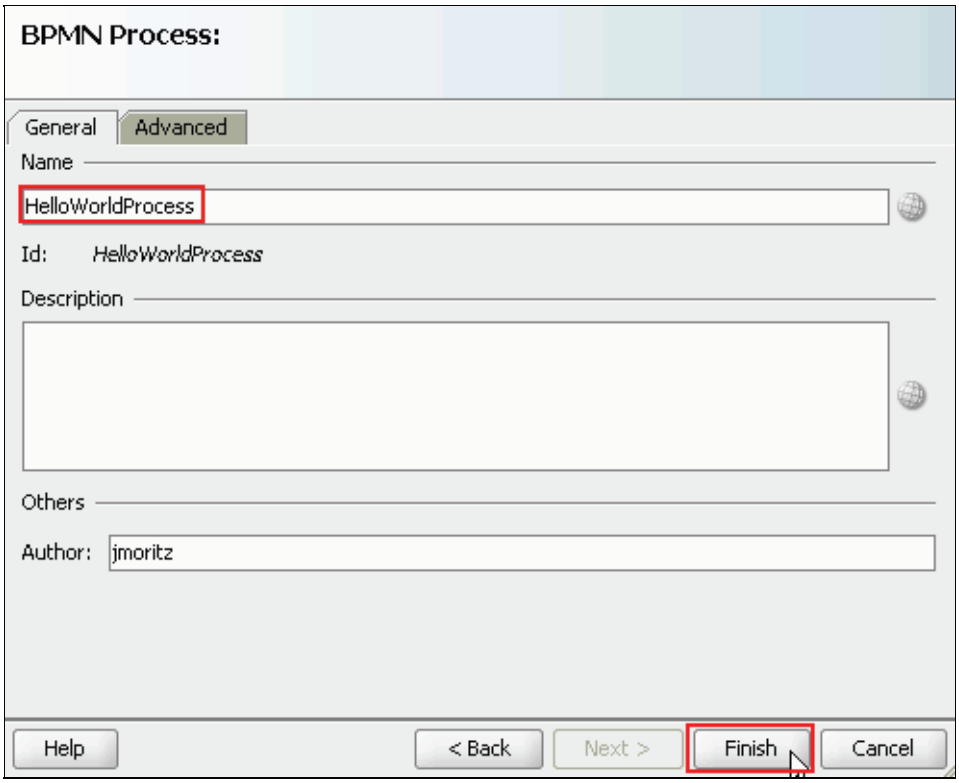
4. To create a new process within this project, first click the **BPM Project Navigator** tab. Then right click on **Processes** and select **New > Process** .



In the **BPM Process** wizard, select the **From Pattern** radio button, and then the **Manual Process** pattern. Click **Next**.



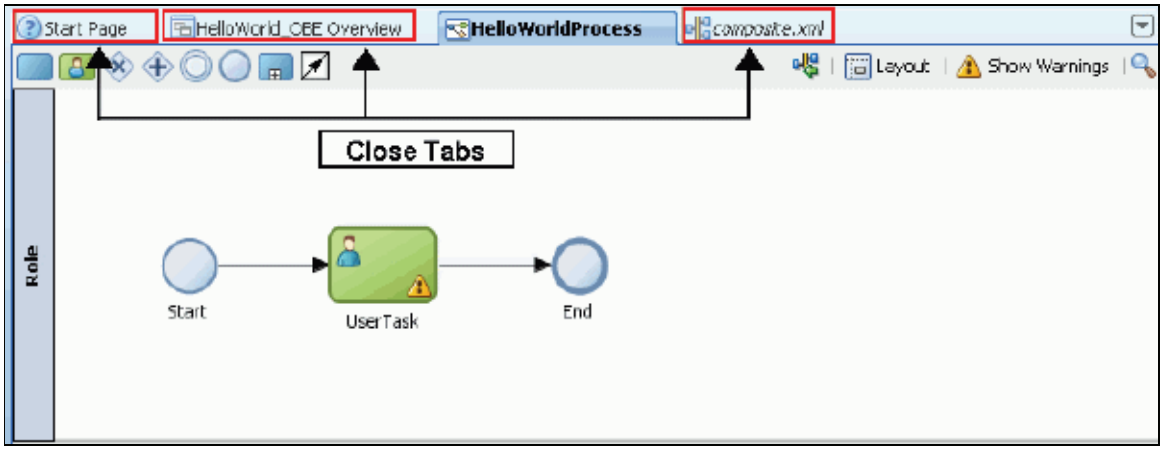
In the next screen, name the process "`HelloWorldProcess`" and click **Finish**.



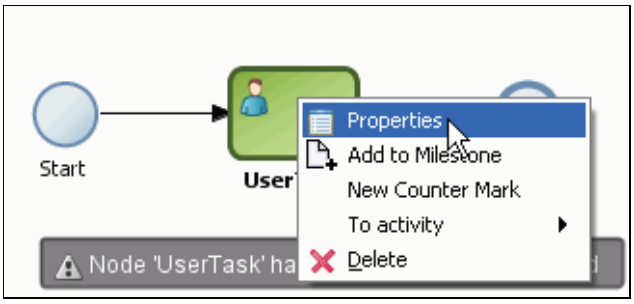
The process model appears in the design editor panel in the middle of the JDeveloper window. The tab name will be same as the name of your new process.

Click the **Save All** icon again.

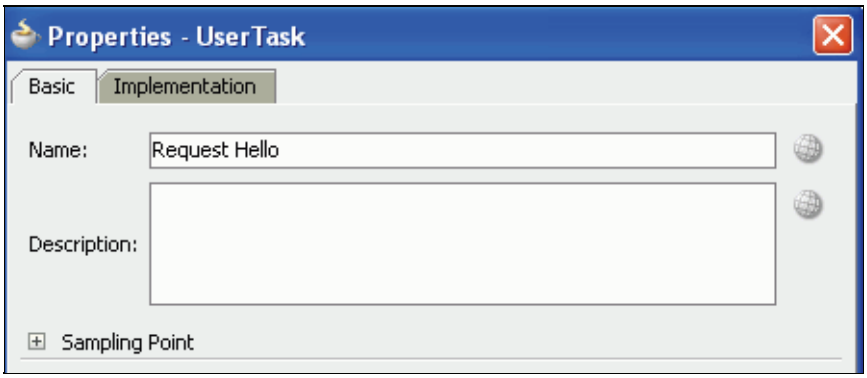
You may wish to close the other tabs, as you will not be using these. An **X** will appear in the upper right corner of the tab when your cursor approaches it. The X will close the tabbed pane. All of these can be easily reopened later from either the menu or one of the navigator panels.



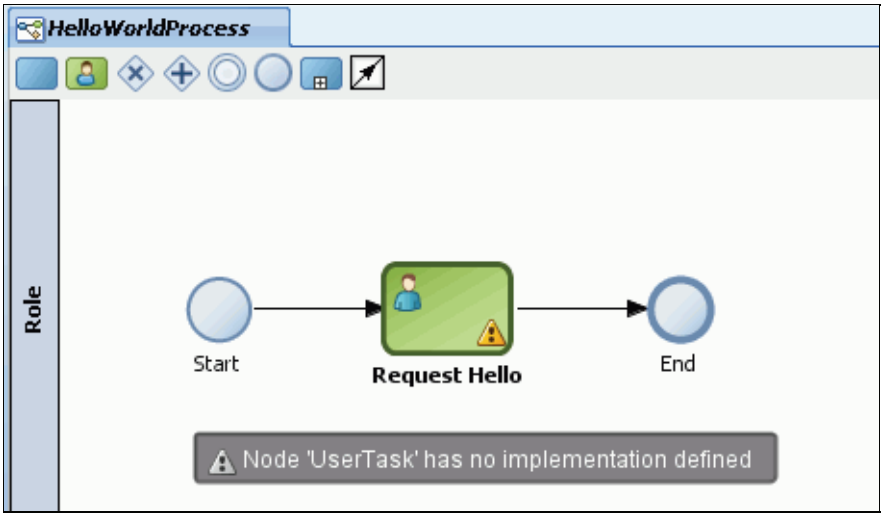
5. Change the name of the user task in the design model. Notice that the model begins and ends with two circular icons. The circle on the left is a **Start** activity and the circle on the right is an **End** activity. Connecting the two circles is a line that represents the flow of activities through the process. This is called the **sequence flow** and sometimes is referred to as the "transition line". Between the Start and End activities is a **User Task** type activity. Right click on this and select **Properties**.



When the **Properties** dialog box appears, on the **Basic** tab, change the name of the activity to "**`Request Hello`**". Click **OK**.



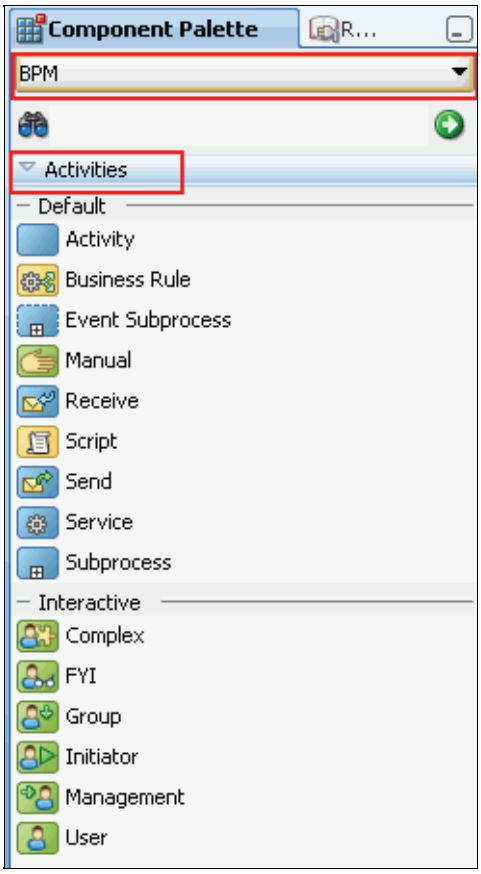
Don't worry about the warning message indicating that no implementation has been defined. You will do this later.



Click **Save All**.

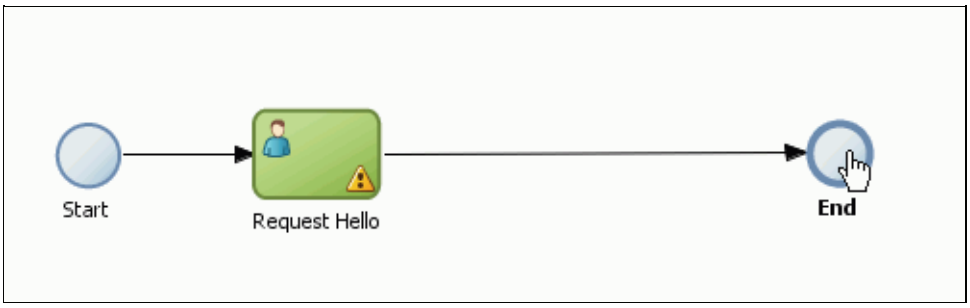
6. Add the Component Palette to the JDeveloper window by selecting **View > Component Palette** from the menu. The palette will appear in the right pane of the window.

   Select **BPM** from the drop-down list at the top of the Component Palette, then expand the **Activities** accordion panel as shown below.



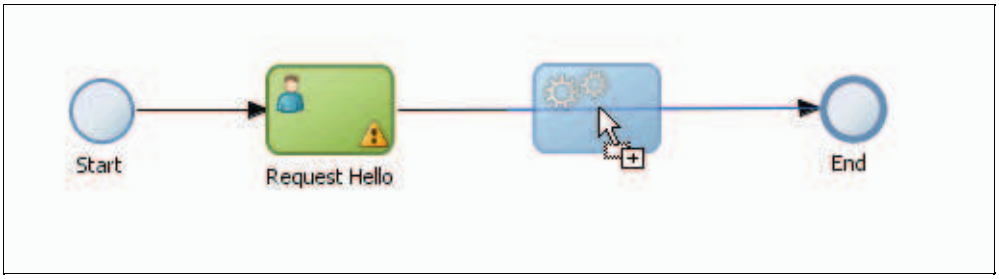
7. Add a service activity to the process. You'll need to first make room for another activity on the sequence flow.

   Click on the **End** activity and drag it to the right, dropping it on the right side of the design panel, allowing enough room for another activity icon to fit between the **Request Hello** activity and the **End** activity.

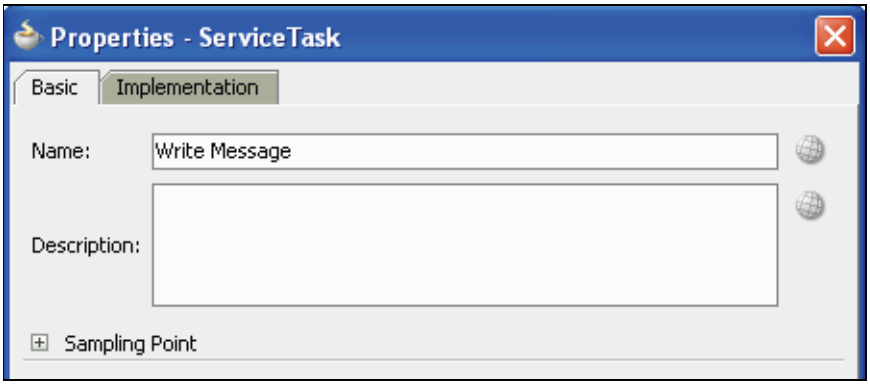


   Now click the **Service** activity icon in the Component Palette and drag it to the Sequence flow between **Request Hello** and **End**. Drop it there. Notice that the transition line turns blue when the drop target area approaches the line.

   Important: The transition line *must be blue* when you drop the object in order for the transition line to be connected to the activity.

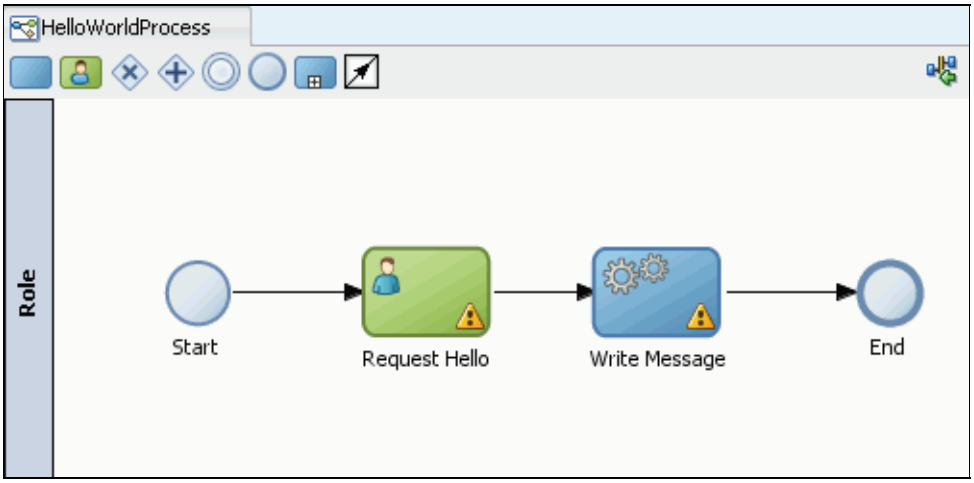


When you drop it, the **Properties** dialog box for the activity opens. On the **Basic** tab, change the activity name to "`Write Message`".



Click **OK**.

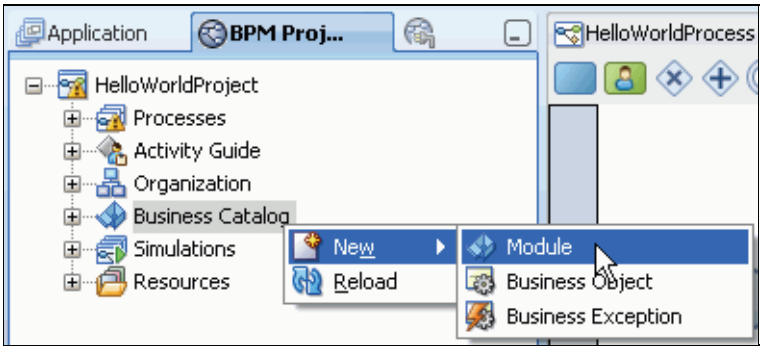
8. Click **Save All**. Your process model should now look similar to this.



Creating the Business Object

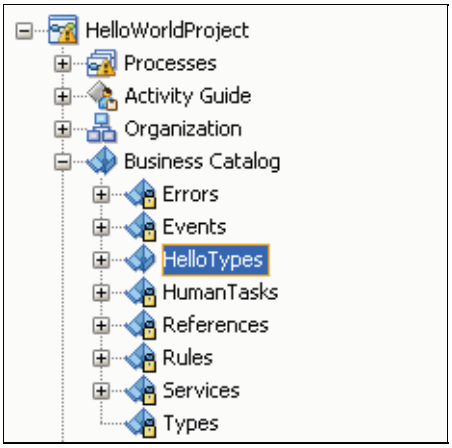
1. Now you will create a business object capable of storing multiple pieces of data, related to the message that the user enters in the Request Hello activity. This object will be populated when the user enters the message. It will then be passed to the Write Message activity so that the message can be written to a file.

   Business objects are stored in *modules* within the Business Catalog. In the **BPM Project Navigator**, expand the **HelloWorldProject** node. Right click on **Business Catalog** and select **New > Module**.

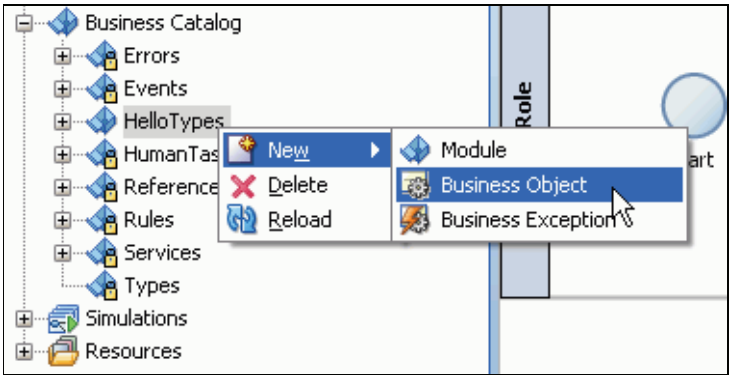


   When prompted to name the new module, enter "`HelloTypes`" and click **OK**.

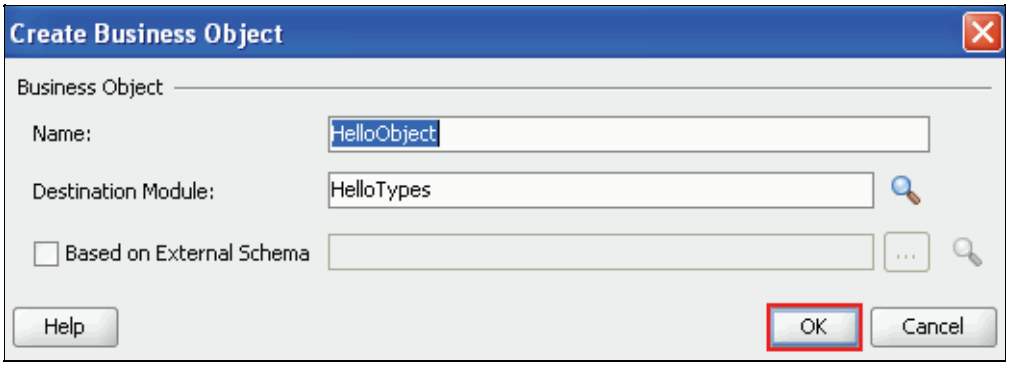
The **HelloTypes** module now appears beneath the Business Catalog node.



2. Right click the HelloTypes module and select **New > Business Object**.



In the Create Business Object window, enter "`HelloObject`" as the Name and accept **HelloTypes** as the Destination Module. Click **OK**.



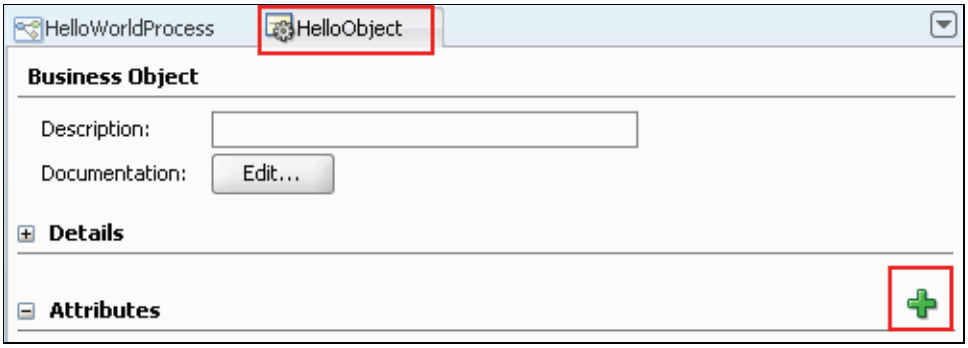
The HelloObject editor now opens in the editor.

3. Add the following three attributes to HelloObject:

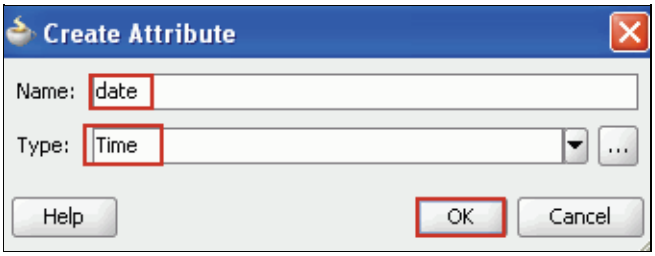
Attribute Name	Type
date	Time
greeting	String
message	String

The following instructions describe how to create the **date** attribute.

Click the plus sign next to the **Attributes** section of the **Business Object** editor as shown below:

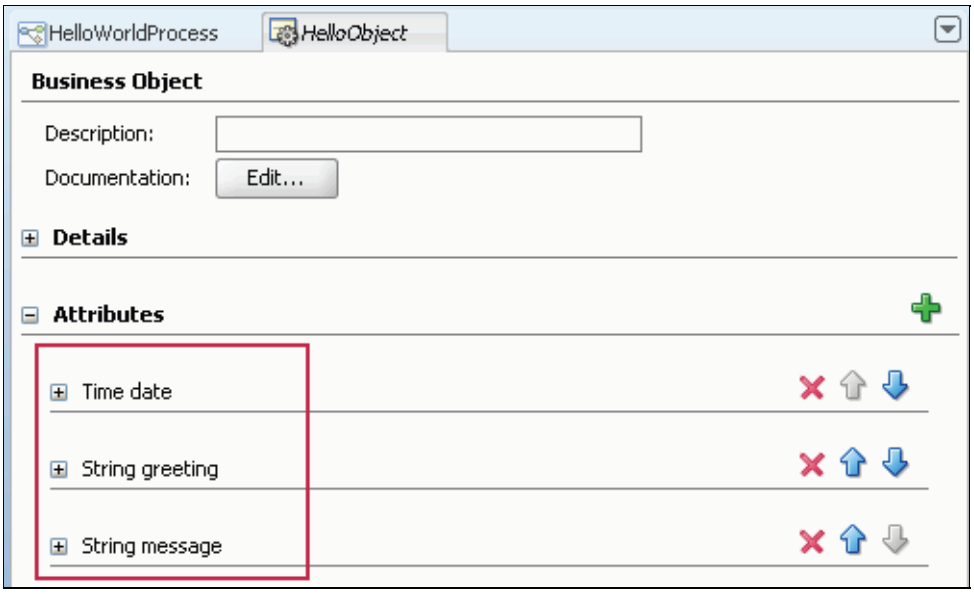


The **Create Attribute** popup appears. Enter "`date`" as the Name value. Select **Time** as the Type from the drop-down list. Click **OK**.



The date attribute now appears in the Attributes section of the Business Object editor.

Continue working in this way to create the other two attributes. When you finish, the Business Object editor should look like this:

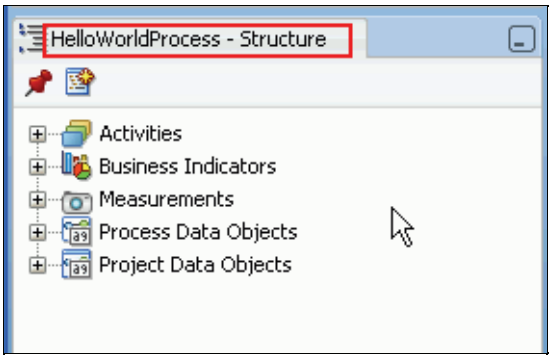


Click **Save All** and close the HelloObject tab in the editor panel.

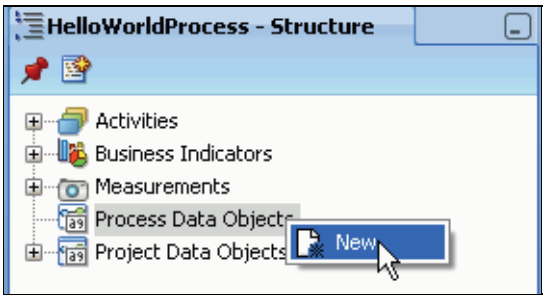
4. Now create a process *data object* of type HelloObject so that you can use it in your process.

   If the **HelloWorldProcess** tab is still open in the editor panel, click anywhere in the design editor to put the focus on the HelloWorldProcess. (If it is not open, select it within the BPM Project navigator by expanding **HelloWorldProject > Processes** to find it)

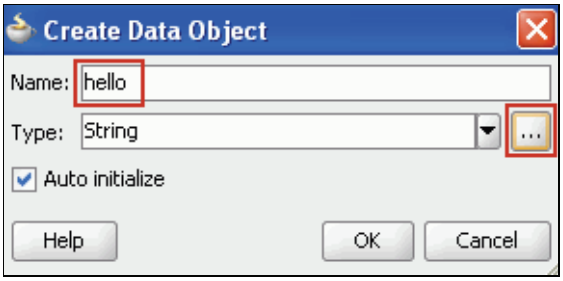
   When a process has been given focus, a detailed outline of its structure appears in the **Structure** pane in the lower left corner of the JDeveloper window.



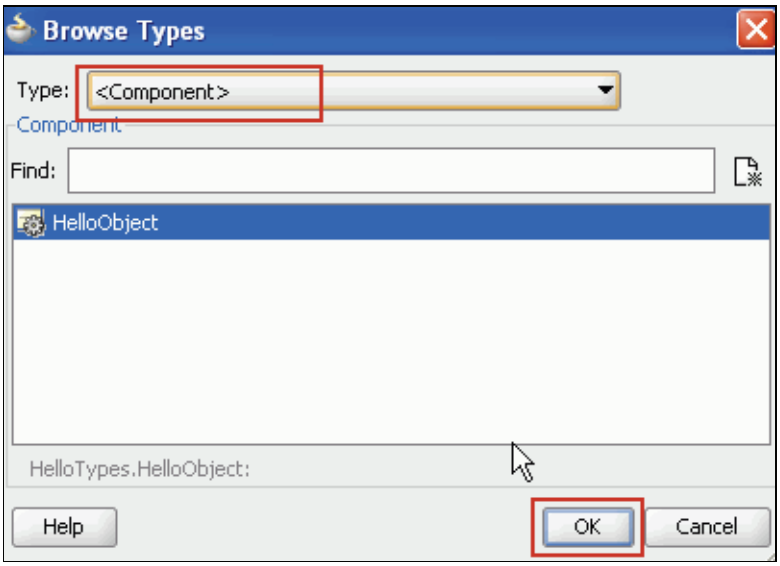
Right click on **Process Data Objects** in the Structure pane and select **New**.



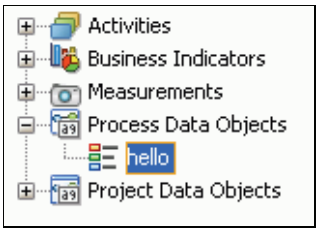
In the **Create Data Object** popup, enter "`hello`" as the Name and click the ellipses button to open another window to search for complex data types.



In the **Browse Types** window, select **<Component>** as the Type and then select **HelloObject** from the list of components appearing below. Click **OK**.



Back in the Create Data Object window, click **OK** again. The `hello` data object now appears in the Structure pane.



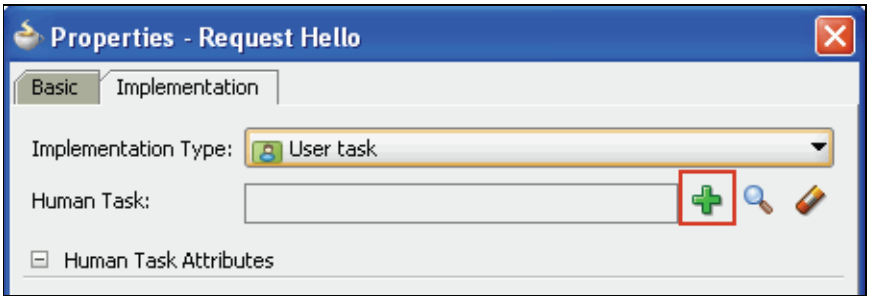
Click **Save All**.

Implementing the User Task

1. Every interactive activity must be bound to a task service to provide its implementation. In the case of a User Task (such as Request Hello), it must be bound to a *Human Task* type task service. You will create the Human Task in this step.

   Right click the **Request Hello** activity in the design editor and select **Properties** to open the Properties window.

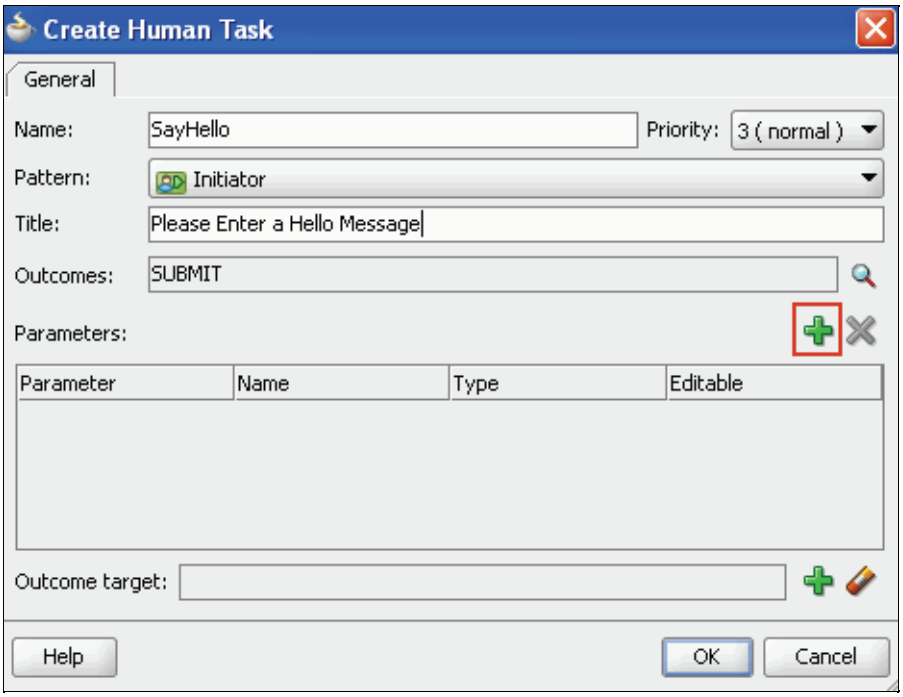
   Click the **Implementation** tab. Next to the **Human Task** field, click the plus sign button as shown below.



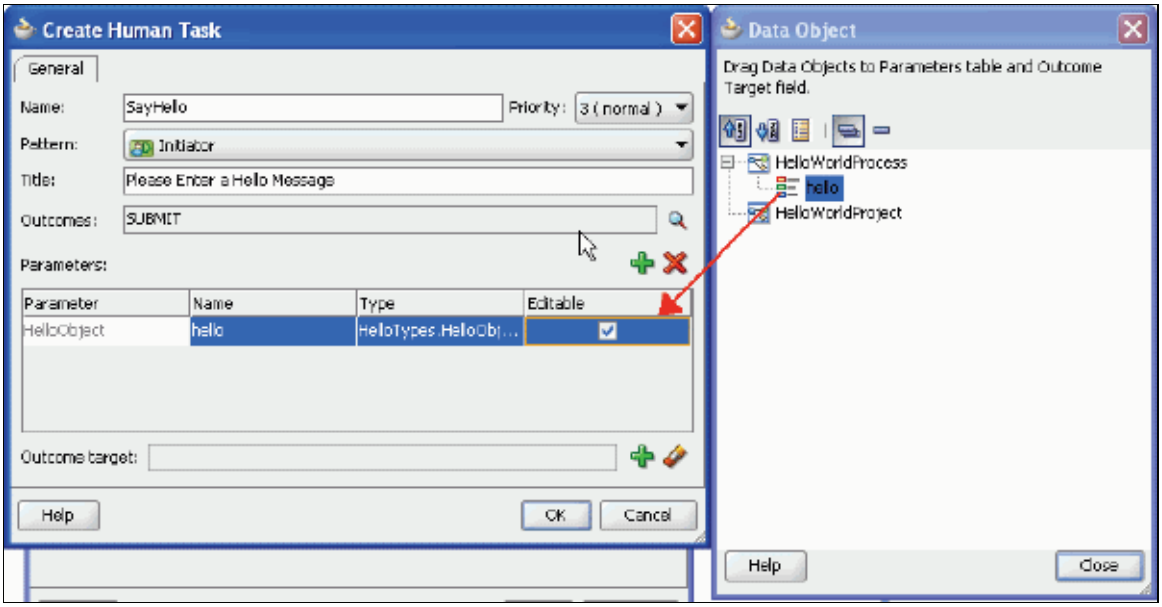
   In the **Create Human Task** dialog box, enter or select the following values for fields in the top portion of the dialog:

Field	Value
Name	`SayHello`
Pattern	**Initiator**
Value	`Please Enter a Hello Message`
Outcomes	Submit *<This is auto selected for you when you choose the pattern>*

   Add a parameter by clicking the plus icon above and to the right of the Parameters panel.

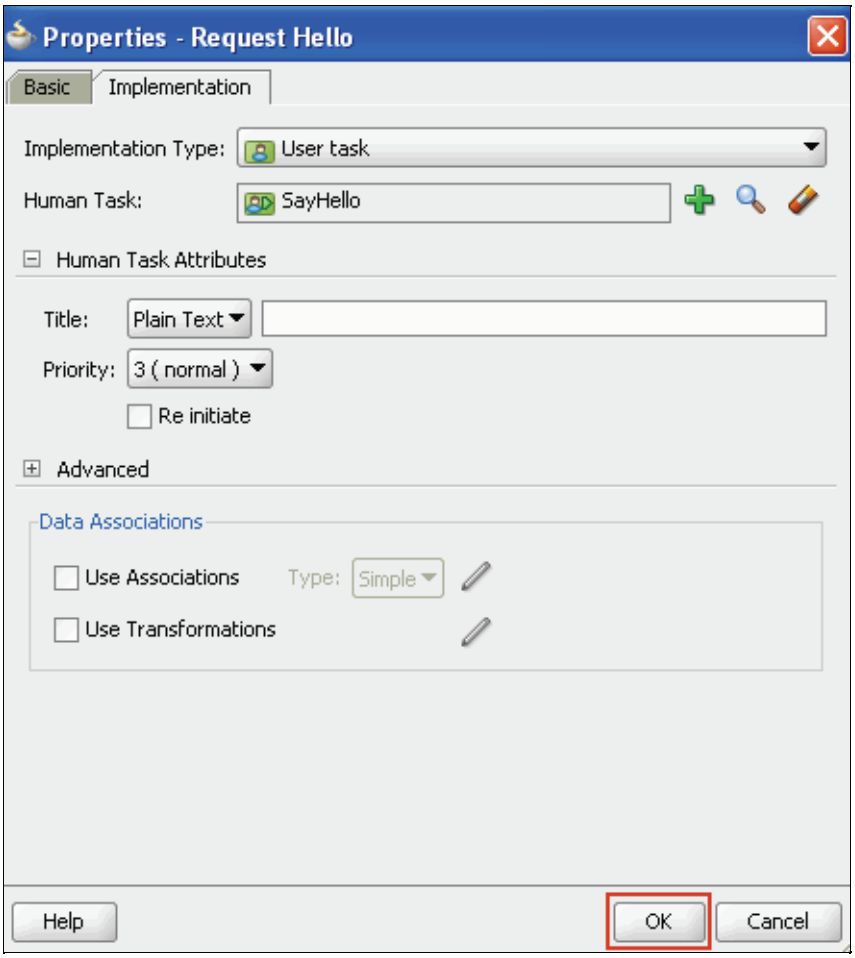


This opens the Data Object window displaying available data objects that you can drag into the Parameters panel. Click on the **hello** data object and drag it into the parameters panel. Select the **Editable** checkbox for the new parameter.



Close the **Data Object** window and then click **OK** in the **Create Human Task** window..

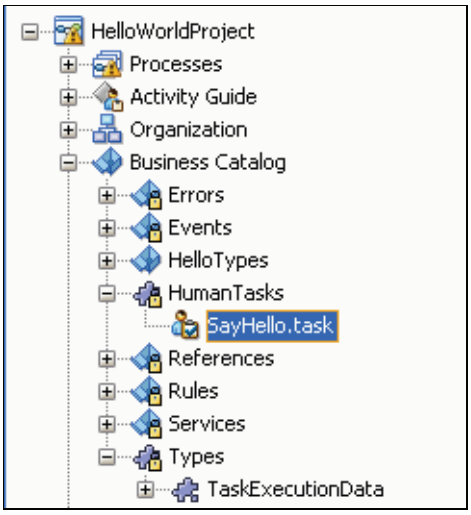
You are returned to the Properties window for the Request Hello activity. Click **OK**.



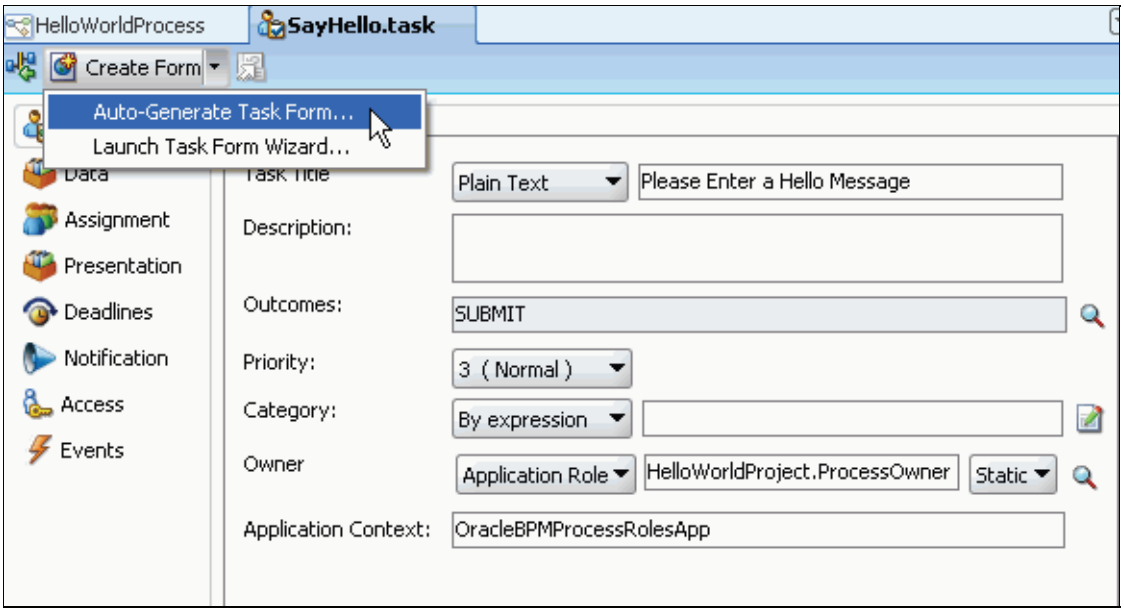
Click **Save All**.

2. Now you must provide a form for the user to enter the Hello message, ensuring that the form is linked to the `hello` data object.

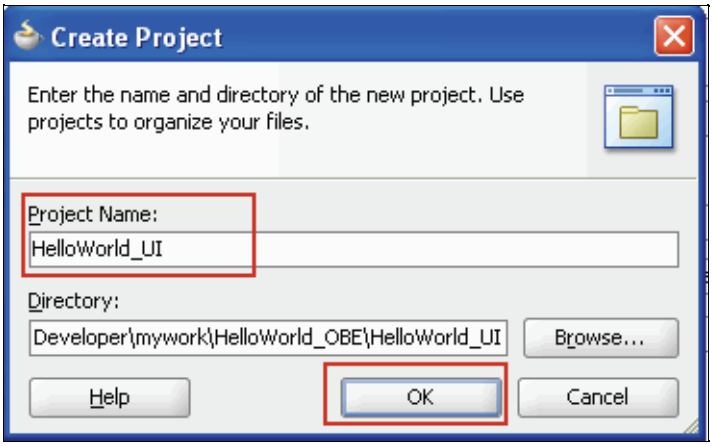
   In the BPM Project Navigator, expand **Business Catalog > Human Tasks**. Here you see the **SayHello.task** object. This is the human task that you just defined.



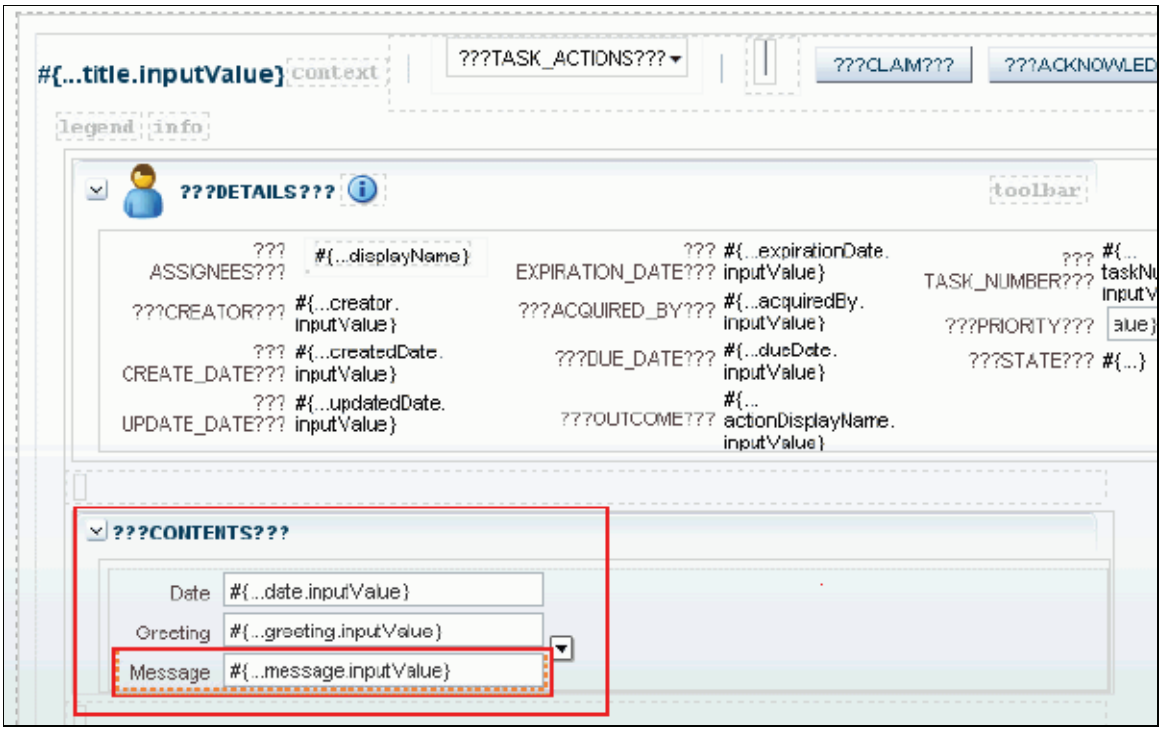
Double click it to open it in the editor. When it opens, click the **Create Form** drop-down list in the editor toolbar on the left and select **Auto-Generate Task Form ...**



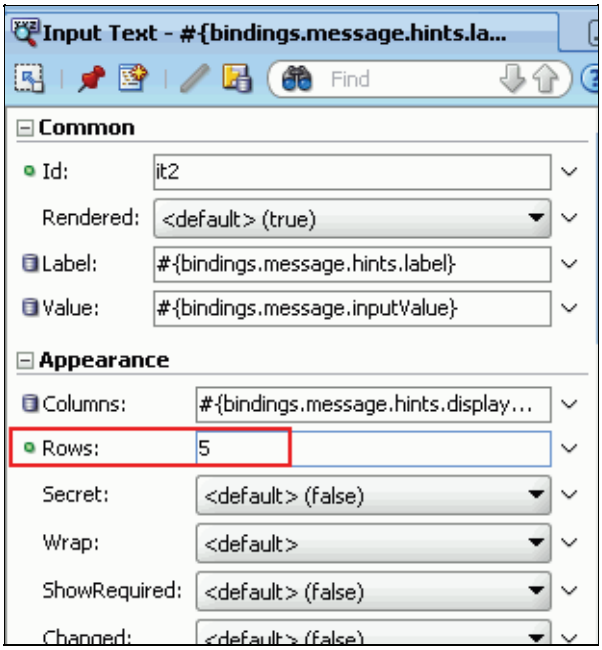
The **Create Project** window opens. It is necessary to create a separate project to contain UI elements (ADF forms). Enter `HelloWorld_UI` as the Project Name and accept the default directory. Click **OK** when finished.



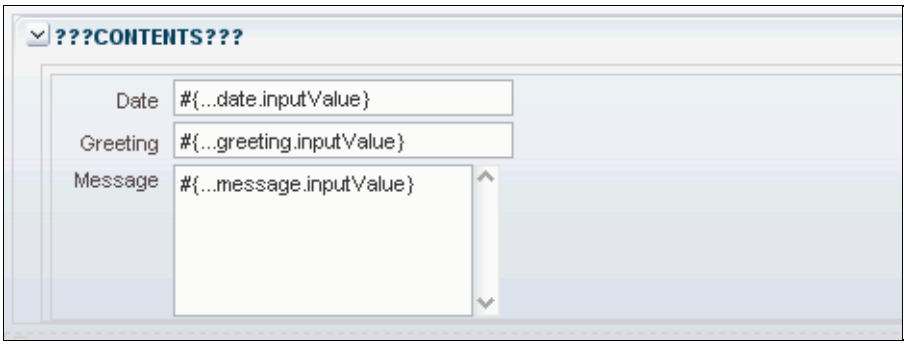
It will take several seconds (even up to a minute, depending on resources) to create the form and open the necessary editor. Eventually, you will see the editor shown below (partial view). The highlighted section is the portion that will be visible to the end user. Notice the **Date**, **Greeting**, and **Message** fields.



Select the **Message** field as shown above. When you do so, the properties for this field appear in a panel in the lower right corner of the JDeveloper window. Expand the **Appearance** accordion panel within this panel and change the value of the **Rows** property to **5**.



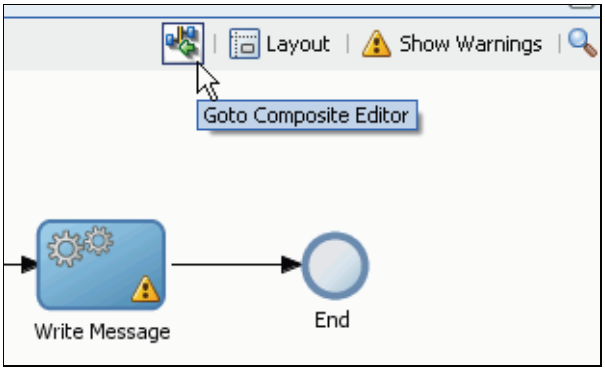
Tab out of the field so that the change will take effect in the form.



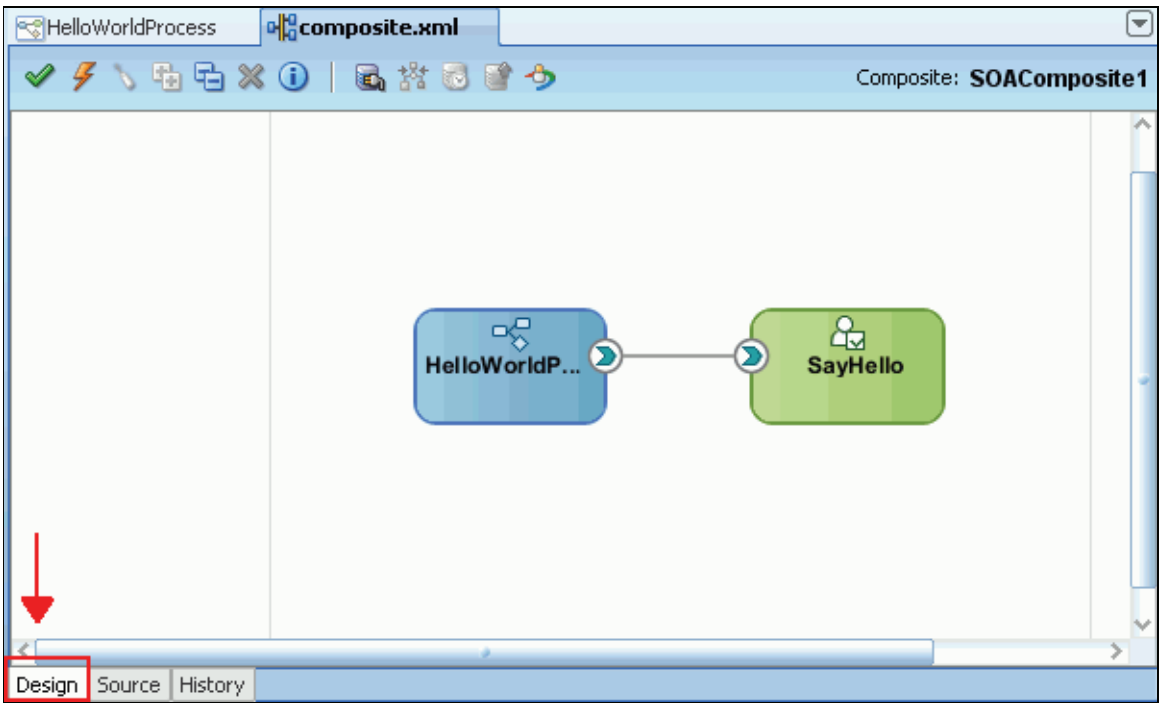
Click **Save All**. Close all tabs in the editor except for the **HelloWorldProcess** tab.

Implementing the File Service

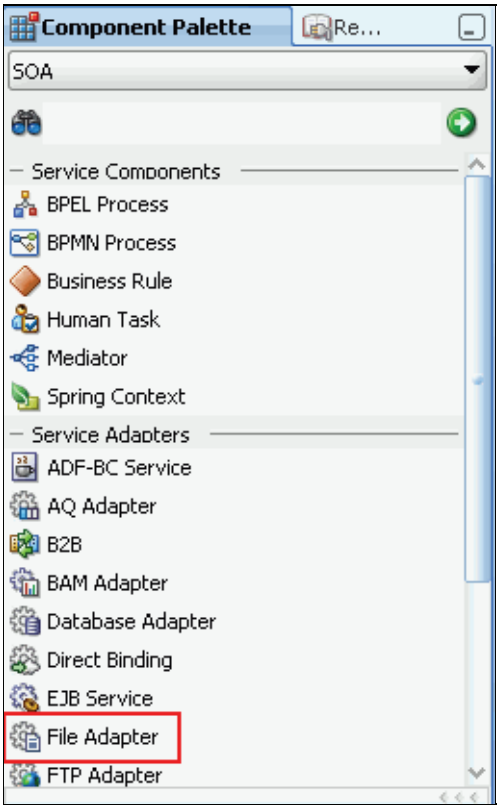
1. Create the implementation for the **Write Message** service activity using the **SOA Composite** editor. Click the **Goto Composite Editor** button on the Design Editor toolbar as shown below.



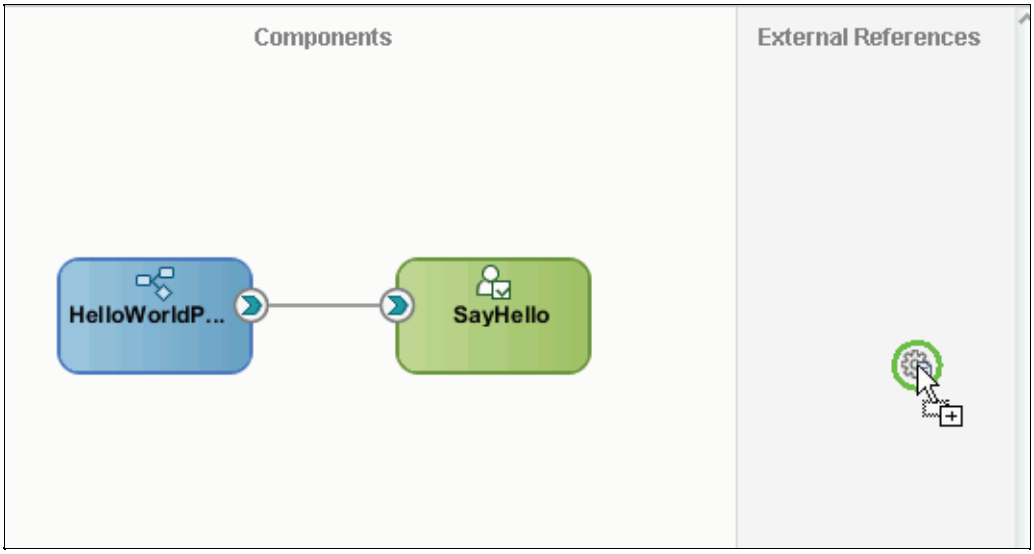
The SOA Composite editor opens. To view the design editor, rather than the XML source, click the **Design** tab at the bottom left margin of the Composite Editor panel. The HelloWorldProcess BPMN component and the SayHello human task component are shown in the composite editor. These are considered components of the SOA composite.



2. Click the **File Adapter** service adapter from the **Component Palette**. Notice that the Component Palette is now showing *SOA* components, by default.



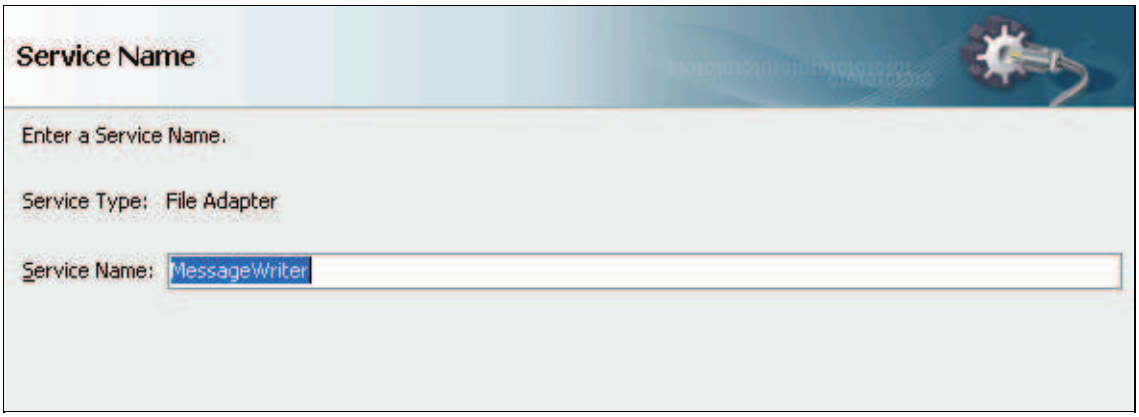
Drag and drop the File Adapter into the External Reference column of the editor.



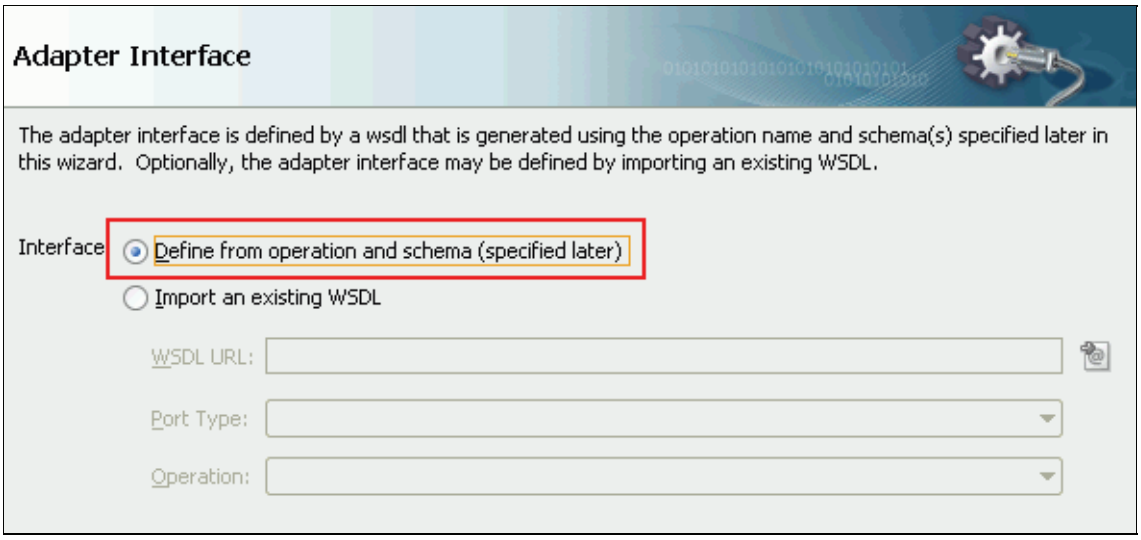
The Adapter Configuration Wizard opens when you drop it.

3. Click **Next** on the **Welcome** page of the **Adapter Configuration Wizard**.

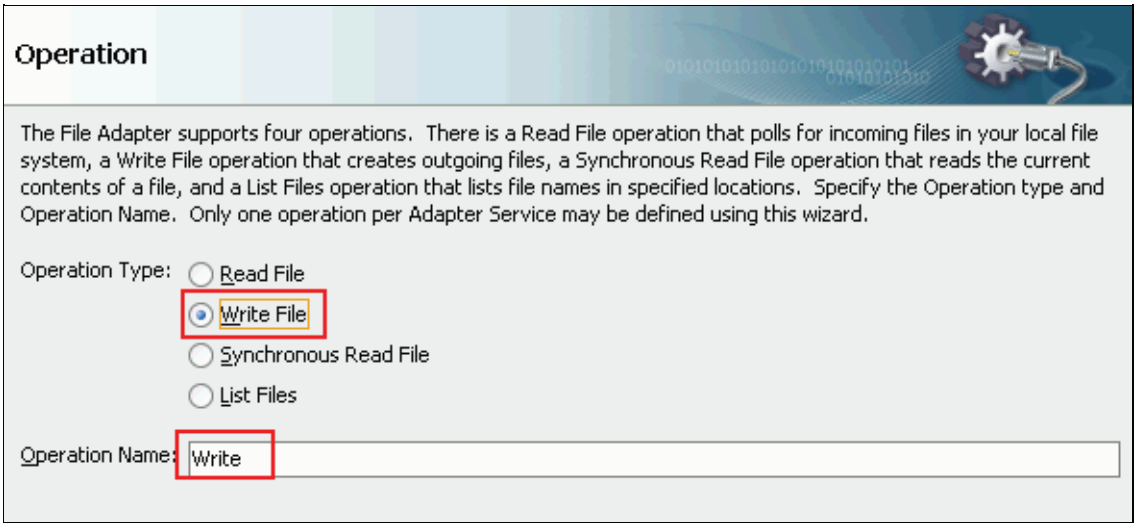
   On the Service Name page of the wizard, name the service `MessageWriter`. Click **Next**.



On the Adapter Interface page, select **Define from operation and schema (specified later)** . Click **Next**.

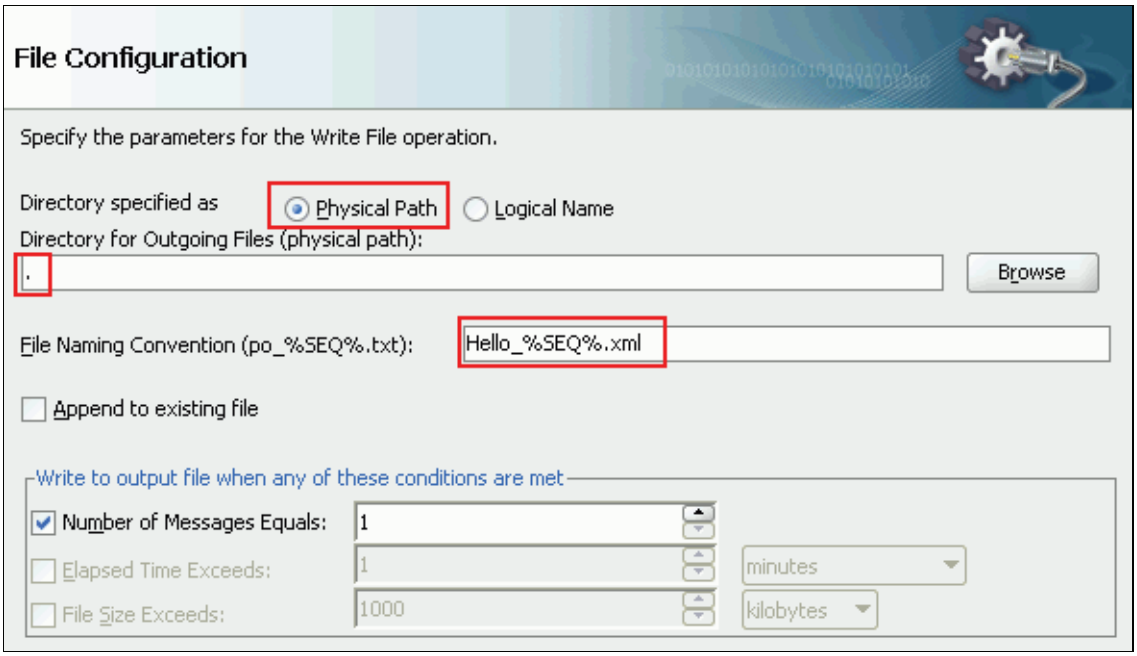


On the Operation page, select **Write File**. The Operation Name will be pre-populated with the name **`Write`**. Accept this value and click **Next**.



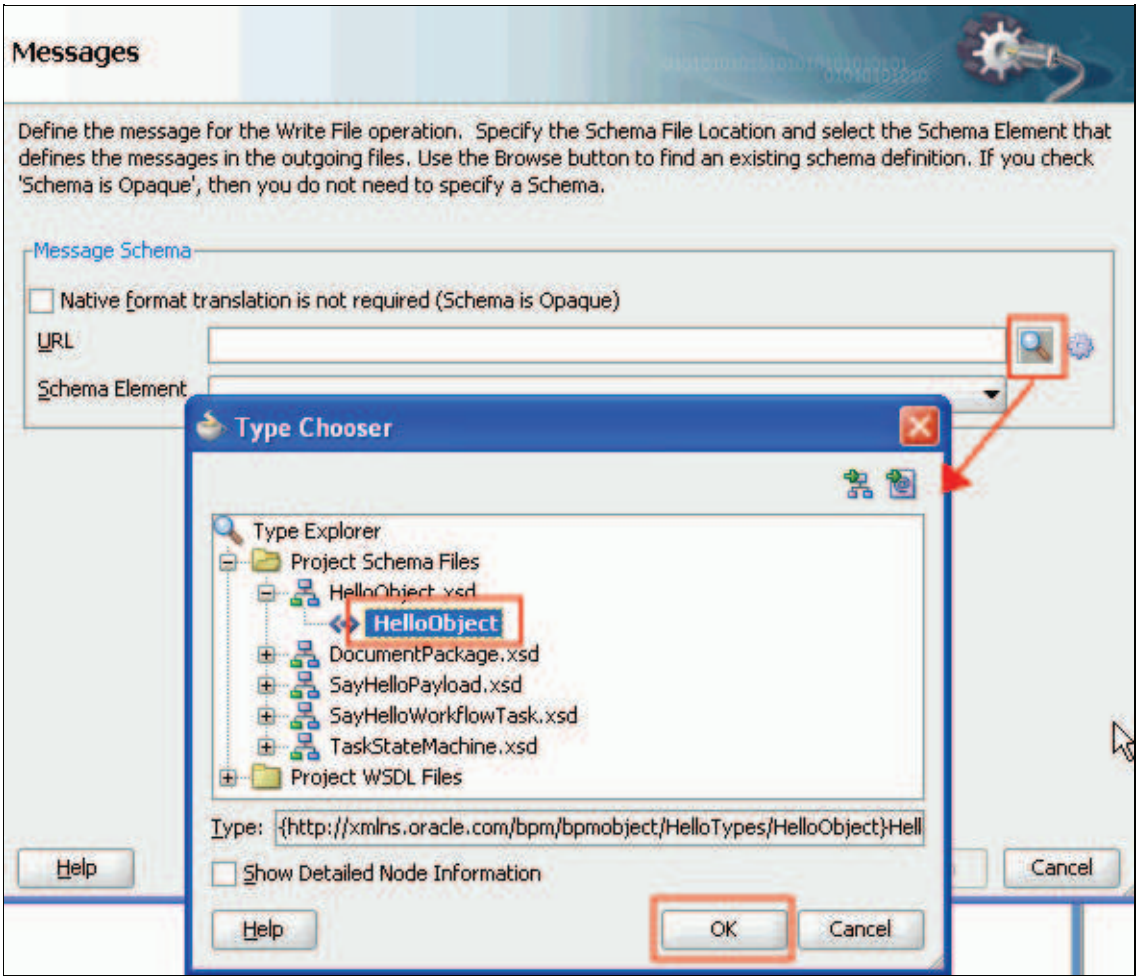
On the File Configuration page, select **Physical Path** as the **Directory specified**. Enter a dot ('**.**') for the **Directory for Outgoing Files (physical path)** field. Also enter a **File Naming Convention** of:

```
Hello_%SEQ%.xml
```

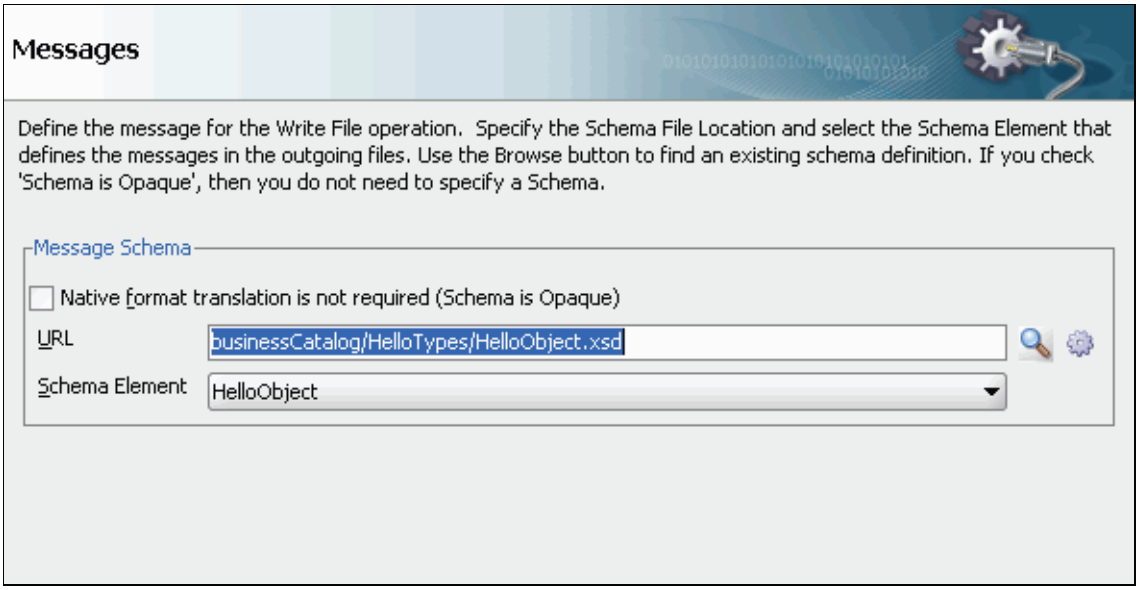


Click **Next**.

In the Messages page, you determine *what* should be written to the file. Click the magnifying glass icon next to the **URL** field to open the Type Chooser popup. Expand **Project Schema Files** and **HelloObject.xsd** to find and select **HelloObject**. as shown below. Click **OK** to accept the selection and return to the Messages window of the wizard.



Click **Next** in the Messages window.



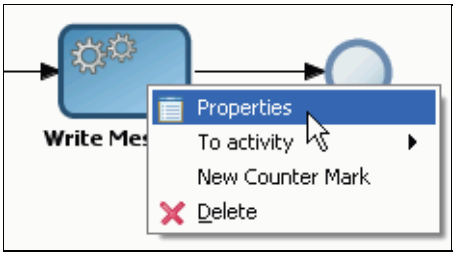
Click **Finish** in the final page of the wizard. The service is created and appears in the SOA Composite editor.

Click **Save All** and close the Composite editor tab.

4. Now you must wire the service implementation you just created to the **Write Message** activity in the BPM process. You need to be in the BPM Process design editor for this.

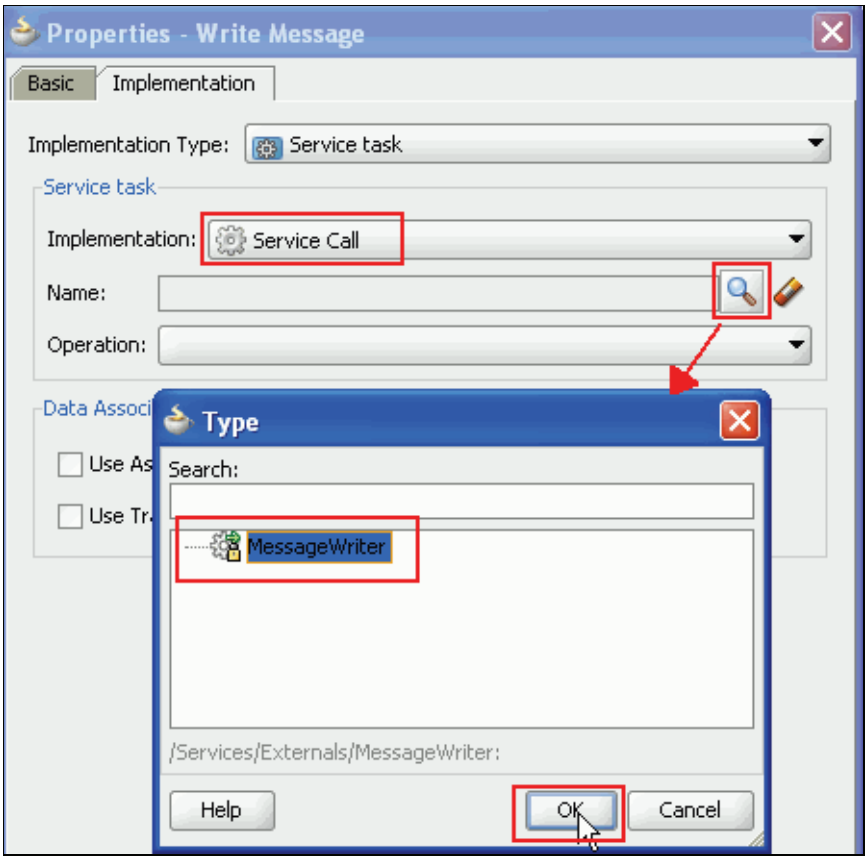
   Note: If necessary, open the **HelloWorldProcess** in the design editor by double clicking on it in the **BPM Project Navigator**.

   Right click on the **Write Message** activity and select **Properties**.



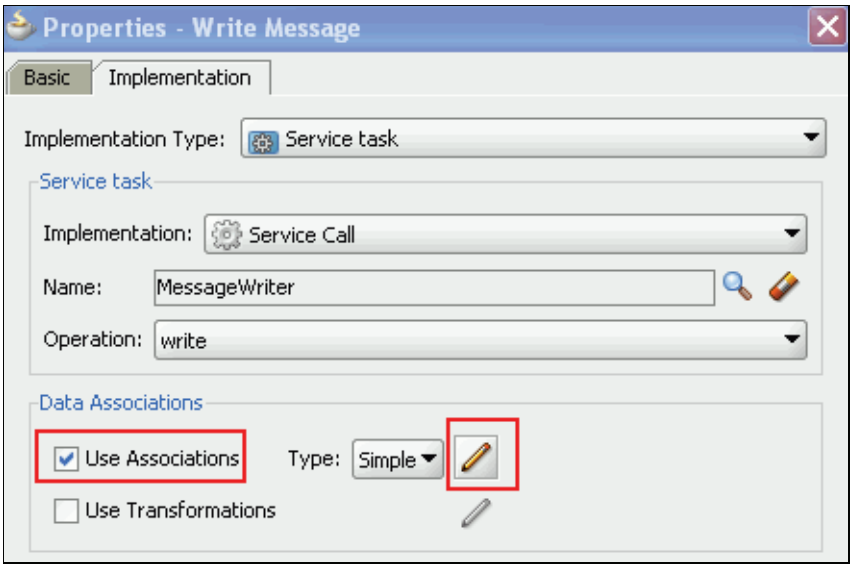
In the Properties window, click the **Implementation** tab.

Select **Service Call** as the **Implementation**. Click the magnifying glass icon next to the **Name** field to browse for a service. The **Type** popup window appears, displaying your **MessageWriter** service. Select it and click **OK.**

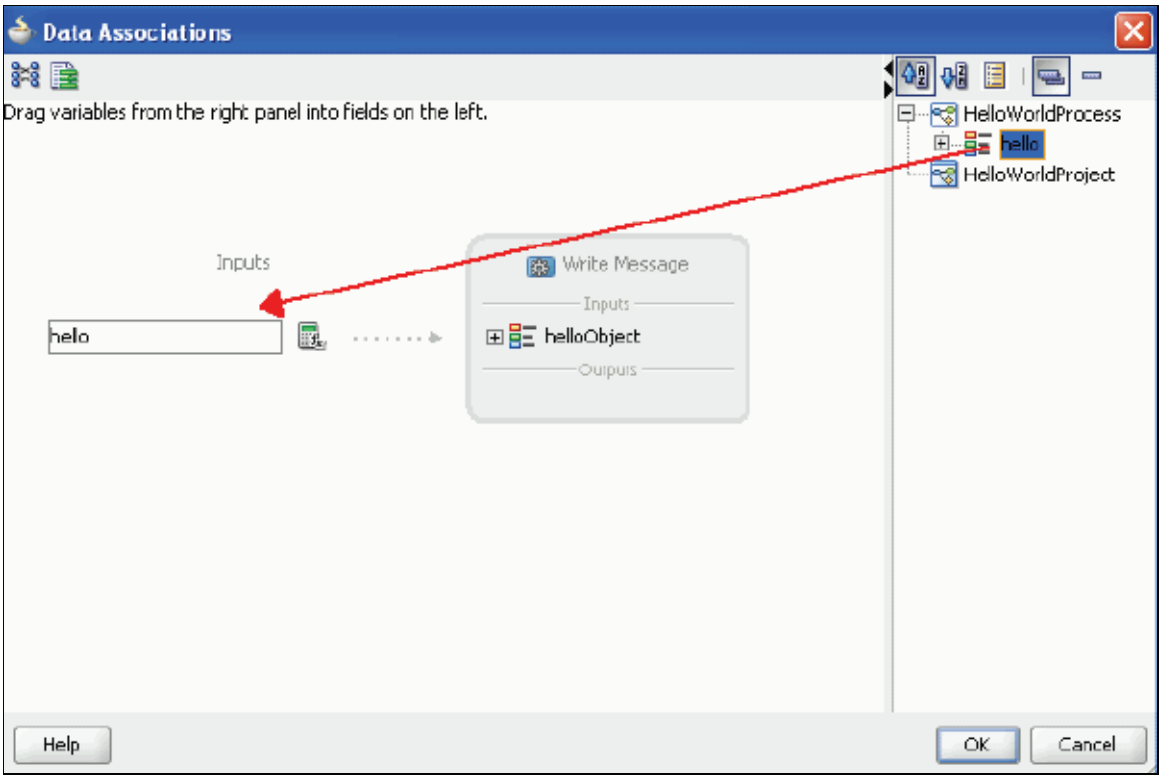


Back in the Properties window, notice that the Operation field has now been automatically set to `write`.

In the **Data Associations** panel, click the **Use Associations** checkbox, then click the pencil icon next to it. This opens an editor that allows you map data into the activity. In this case, you want to map the *`hello`* object into the activity so that the MessageWriter service can write the message.



In the **Data Associations** window that opens next, drag the *`hello`* data object from the right column over to the **Inputs** field on the left side of the window. Notice that this maps the *`hello`* data object to the *`helloObject`* expected as input to the activity's service implementation.



Click **OK** in the Data Associations window to save the mapping and return to the Properties window.

Click **OK** in the Properties window.

Click **Save All**.

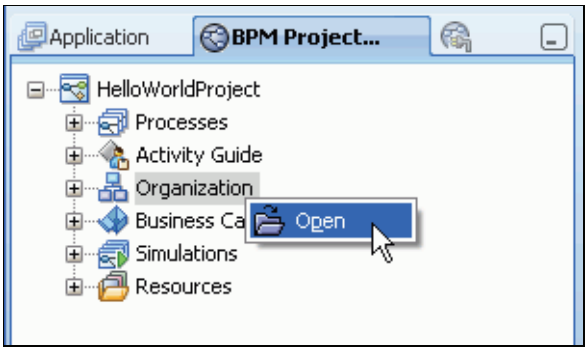
Enhancing the Basic Hello World Process

In this section, you enhance the basic process you just built by adding a review capability for the message entered by the user. You add another business object to hold review-related data and use this business object in conjunction with a business rule that tests the length of the greeting and message. You also add another human interaction that allows the user with the Reviewer role to review the entered message and accept or reject the message. Then you change the process model, itself, so that it conditionally branches to the Review Message activity and potentially back to the Request Hello activity if the message was rejected. A script task will also be needed in order to initialize a variable used in the conditional branching logic.

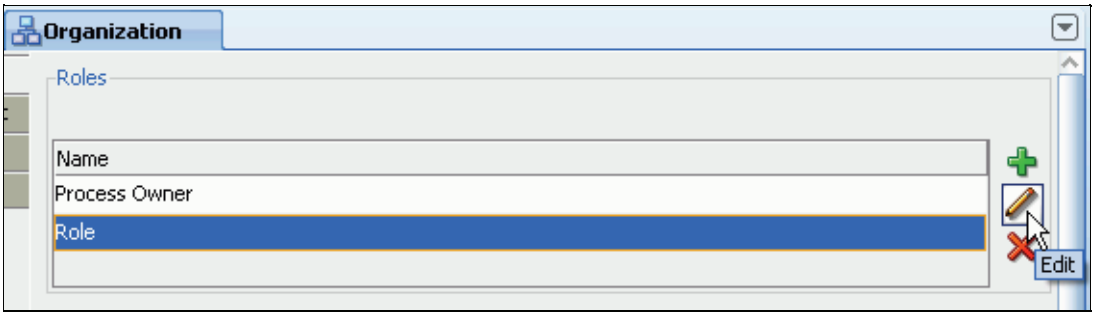
You begin the enhancement by changing the name of the role assigned to the user who executes the Request Hello activity to something more meaningful. Then you add a new role for the user who reviews the message.

Adding the ReviewNeeded Business Object

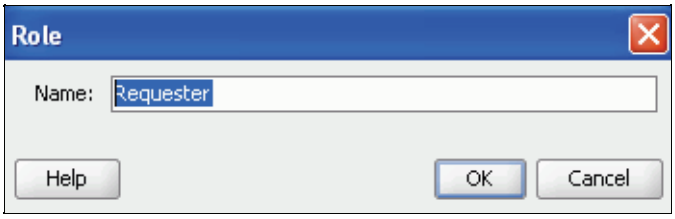
1. Change the name of the **Role** role. In the BPM Navigator, right click on the **Organization** node beneath **HelloWorldProject** and select **Open**.



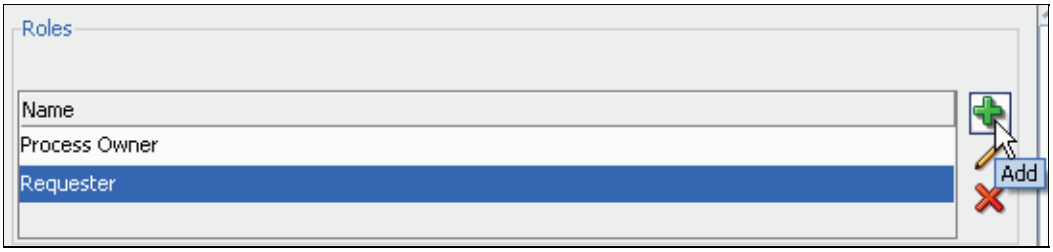
The Organization editor opens in the center panel. Select the **Role** role and click the pencil icon to edit it.



The Role popup window appears. Enter the name `Requester` and click OK.

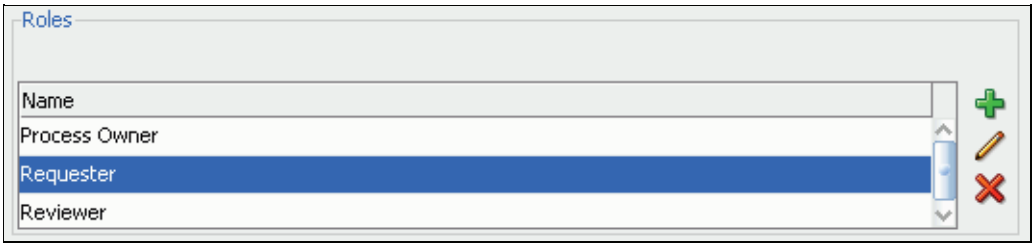


2. With the Organization editor still open, click the green plus sign button to add a new role.



Enter the name **Reviewer** in the Name field of the Role popup window and click **OK**.

Your Organization editor should now look like this:

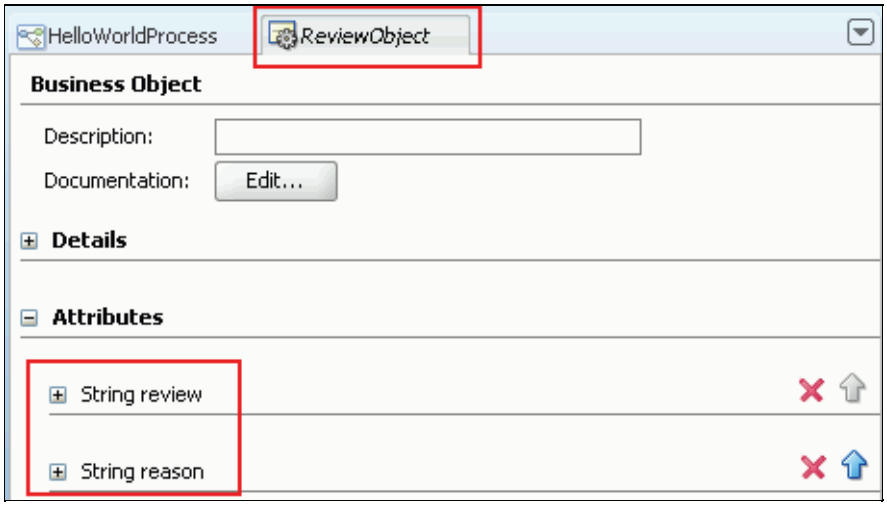


3. Click **Save All** and close the Organization editor.

4. Add a new business object in the same manner that you did in the **Creating the Business Object** section above. Follow Steps 2 and 3 of that section, placing the new business object in the **HelloTypes** module and substituting the name and attributes mentioned below.

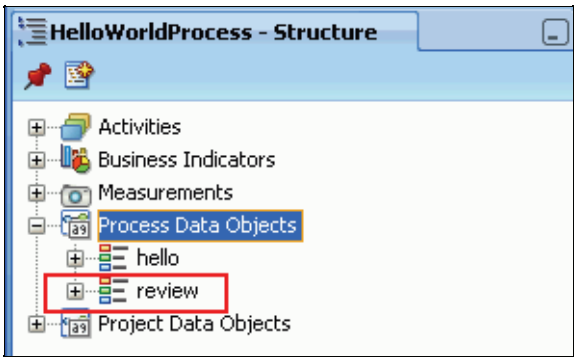
   Name the object `ReviewObject` and add the following attributes to it:

Attribute Name	Type
review	String
reason	String

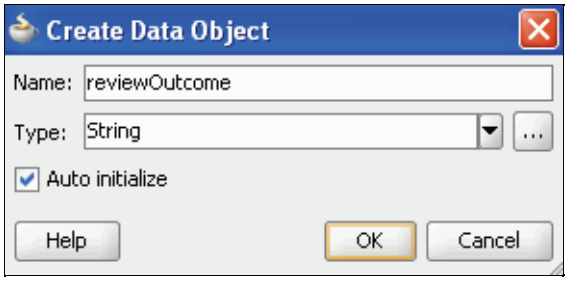


   Click **Save All** and close the object editor. The ReviewObject should now appear in the BPM Navigator beneath the HelloTypes folder.

5. Declare a process data object of type **ReviewObject** in the same manner that you did in the **Creating the Business Object** section above. Follow the instructions in Step 4, substituting `review` as the data object name.



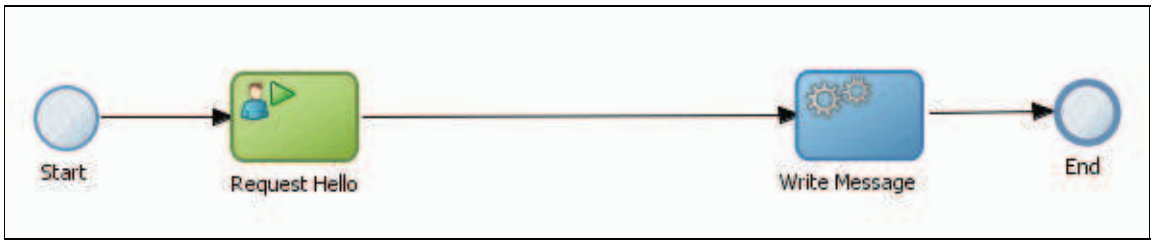
6. Add one more process data object of type String, called `reviewOutcome`. Follow the same procedure as the previous step, except choose String as the type.



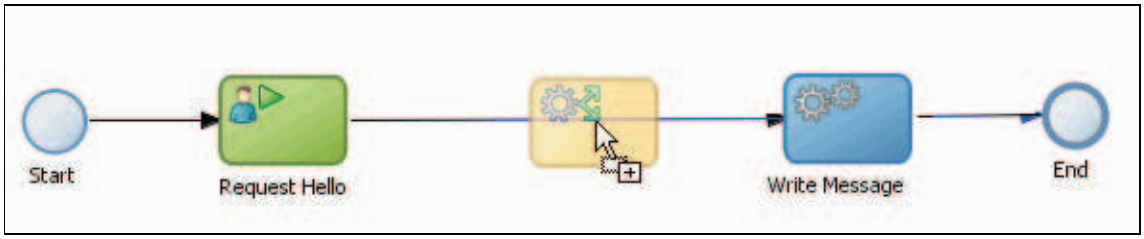
Adding a Business Rule

1. Add a business rule to the design model.

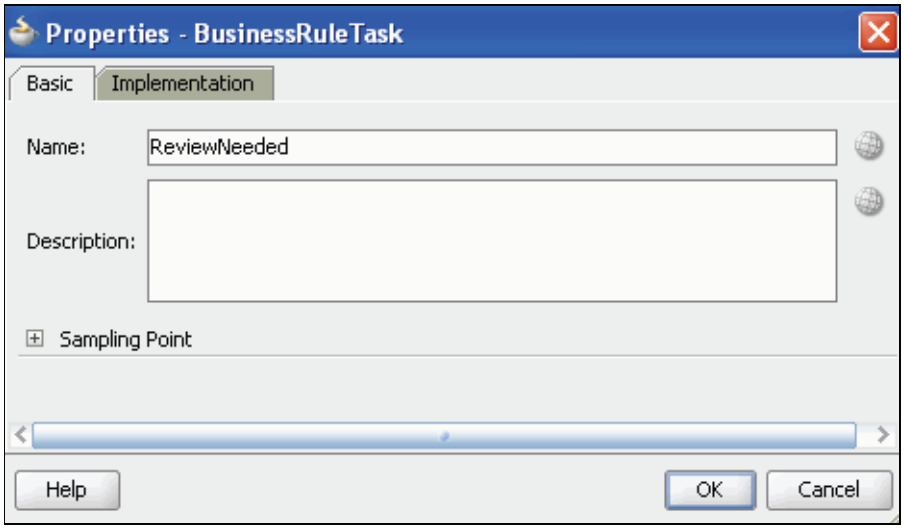
Begin by making more room on the sequence flow between the Request Hello and Write Message activities.Move the End activity and the Write Message activity further to the right.



Expand the **Activities** accordion panel in the Component palette, then click and drag a **Business Rule** into the design editor, dropping it onto the sequence flow between **Request Hello** and **Write Message**.



The Properties dialog for the business rule appears when you drop the object. Enter **`ReviewNeeded`** in the **Name** field. Click **OK**.



Click **Save All**.

2. In the next step, you will define the implementation of the ReviewNeeded business rule using the Composite editor. In order to make the two business objects available to the Composite editor, you must copy the **`xsd`** files that were generated for each object when you created them from the `businessCatalog\HelloTypes` folder to the `xsd` folder within the directory structure of your project. *[This is a workaround to a known bug in this release of the product]*

Open Windows File Explorer and navigate to:

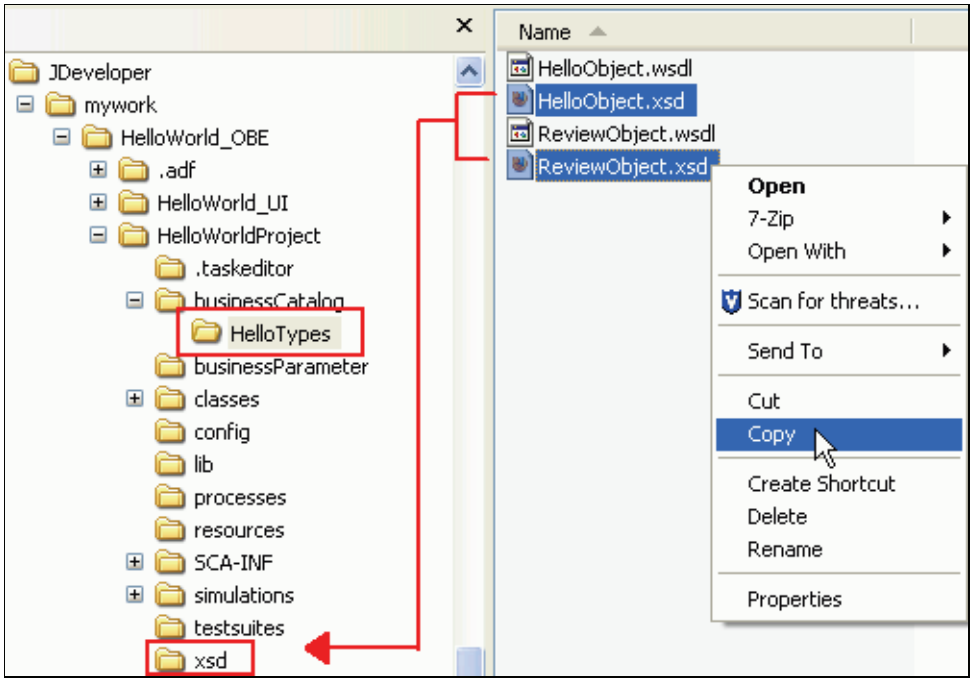
```
C:\JDeveloper\mywork\HelloWorld_OBE\HelloWorldProject\businessCatalog\HelloTypes
```

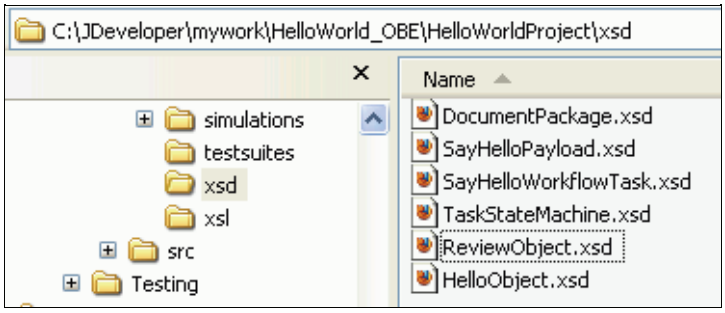
In this folder, find and copy the following two xsd files:

```
HelloObject.xsd
ReviewObject.xsd
```

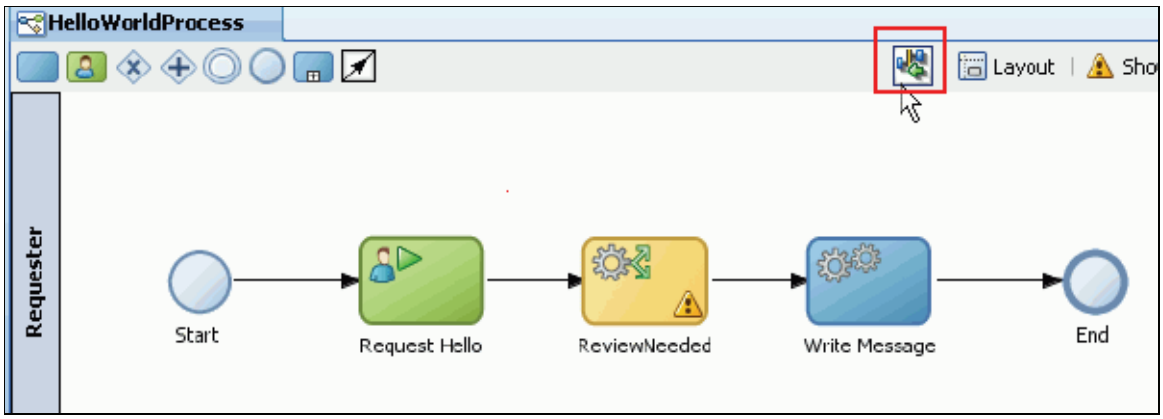
Paste them into the following folder:

```
C:\JDeveloper\mywork\HelloWorld_OBE\HelloWorldProject\xsd
```

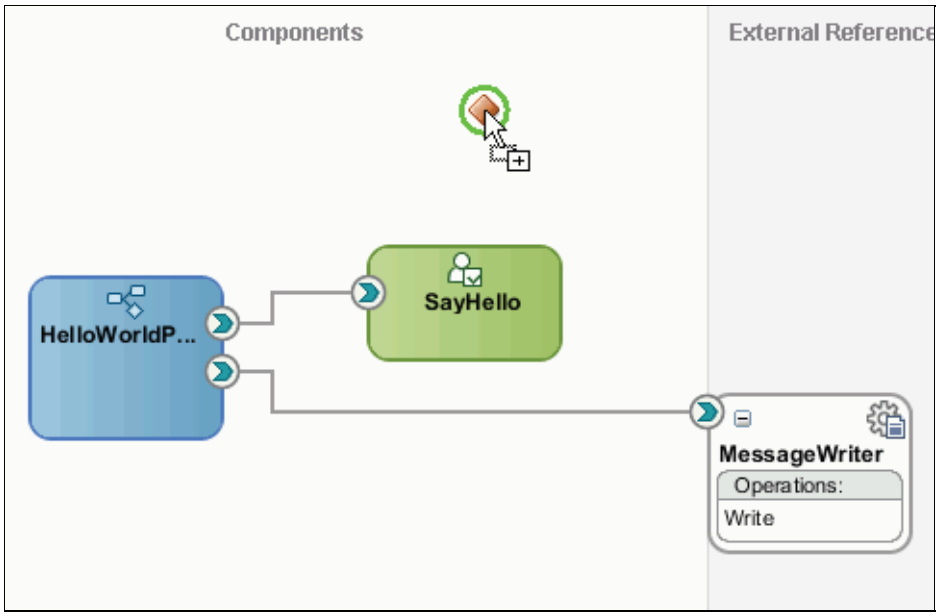




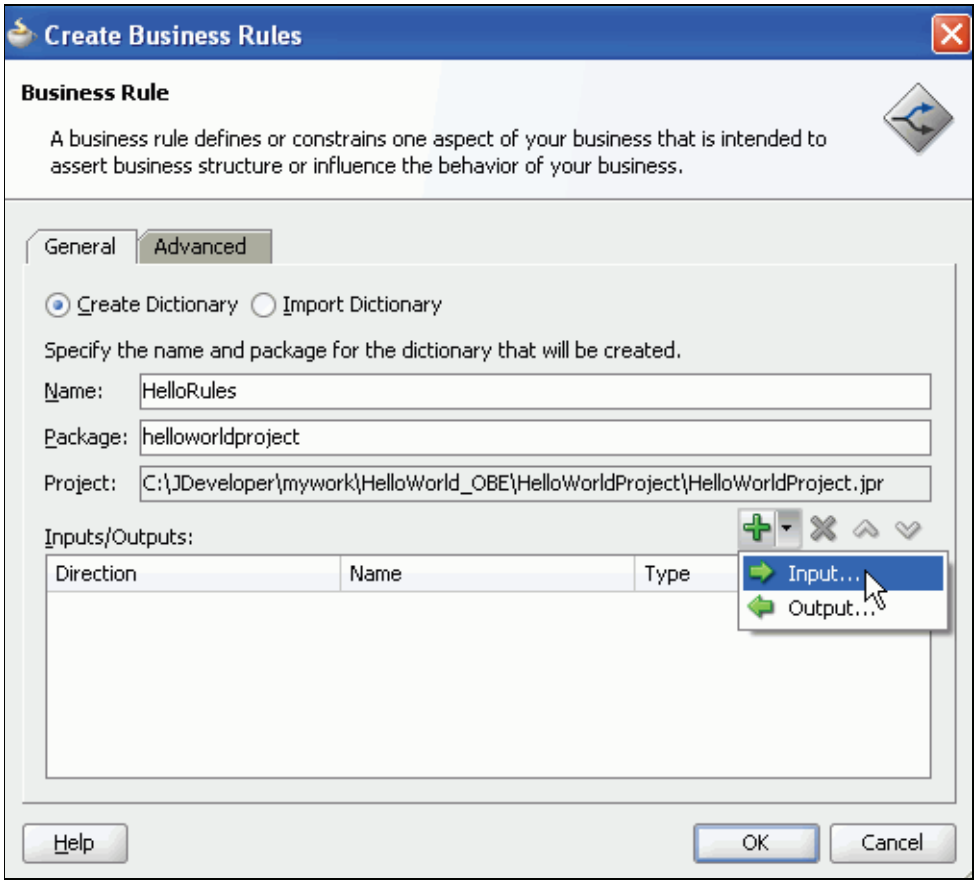
3. Open the Composite editor by clicking the **Goto Composite Editor** button on the Design editor toolbar.



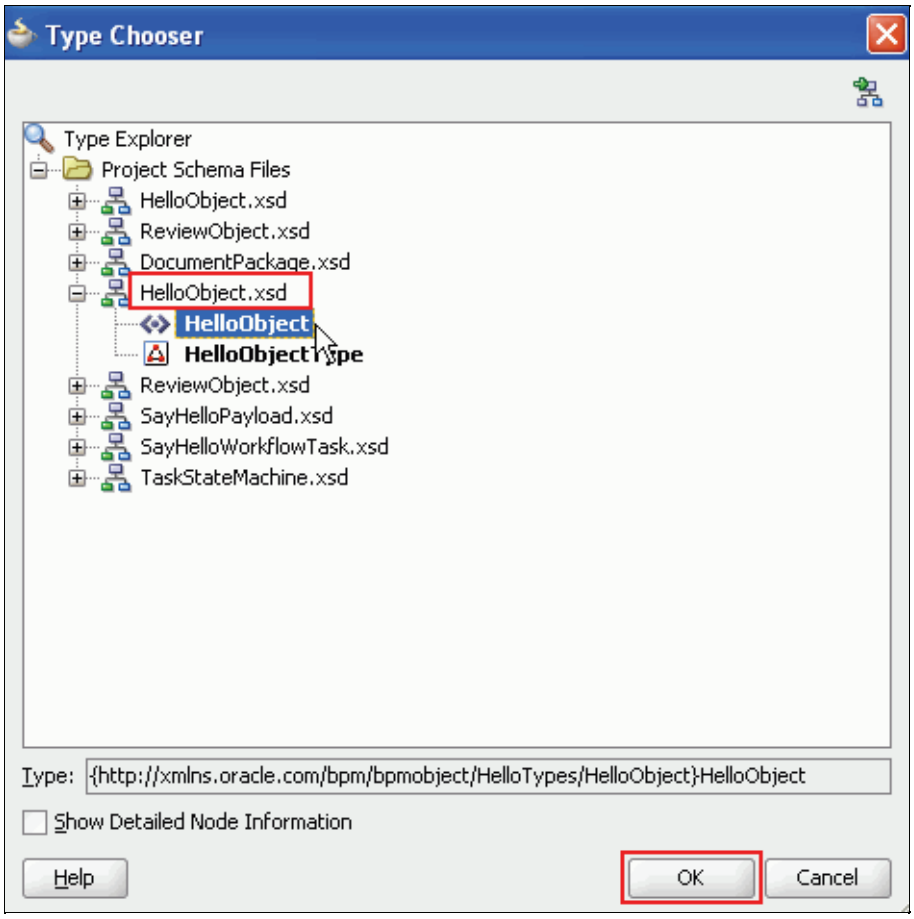
4. Add a **Business Rule** component to the Composite Editor. From the Component palette panel on the right, click and drag a Business Rule from the Service Components section of the palette and drop it in the Components column of the Composite Editor.



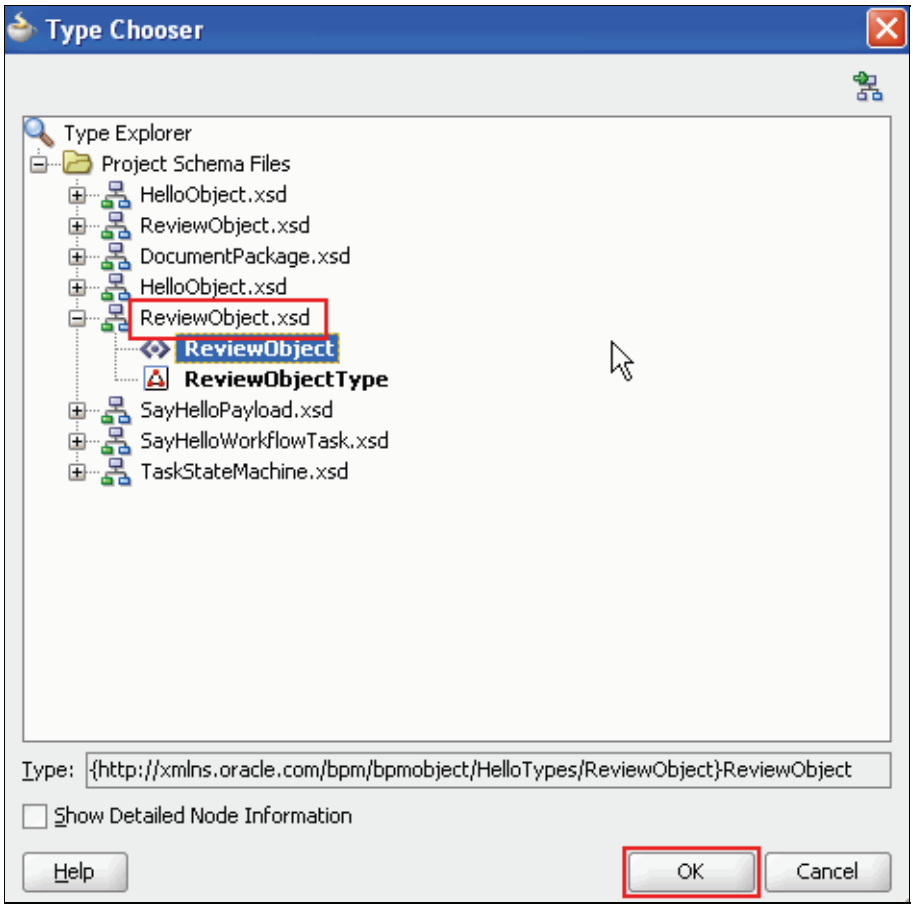
When you drop the component, the Properties dialog appears. Enter **`HelloRules`** in the Name field. Then click the green plus button above the **Inputs/Outputs** section and select **Input...**



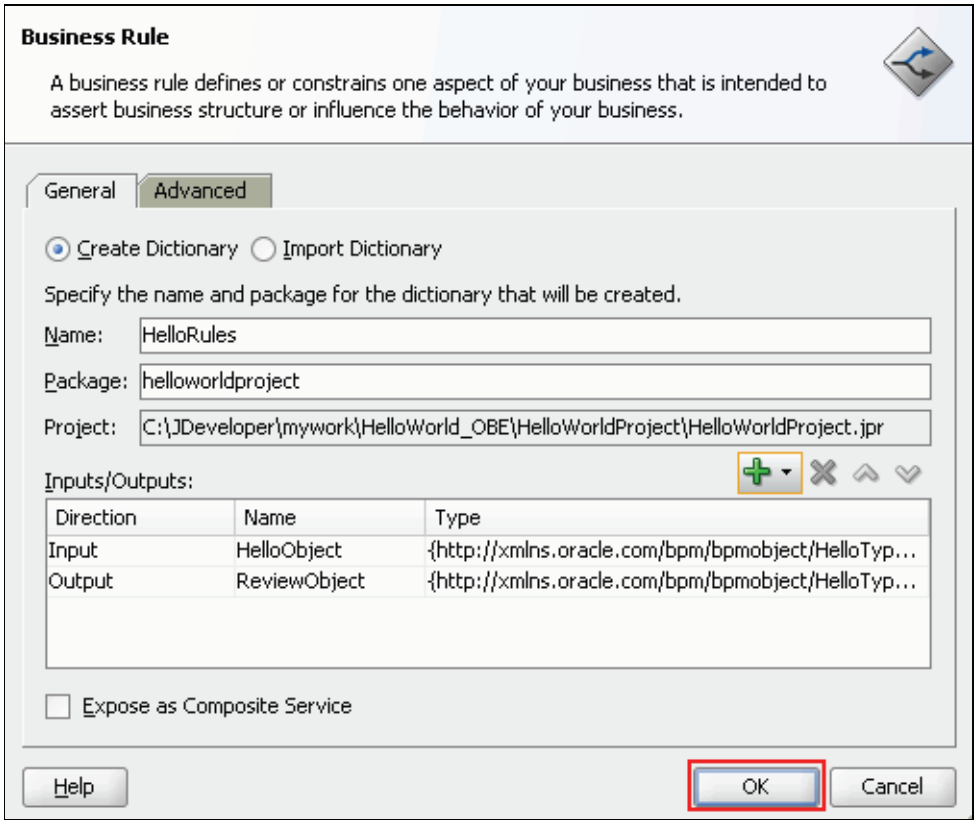
The **Type Chooser** popup window appears. Expand the second **HelloObject.xsd** entry, then select the **HelloObject** beneath it and click **OK**.



Click the green plus button once more and this time select **Output...** When the Type Choose popup window appears, expand the second **ReviewObject.xsd** entry and select **ReviewObject**. Click **OK**.



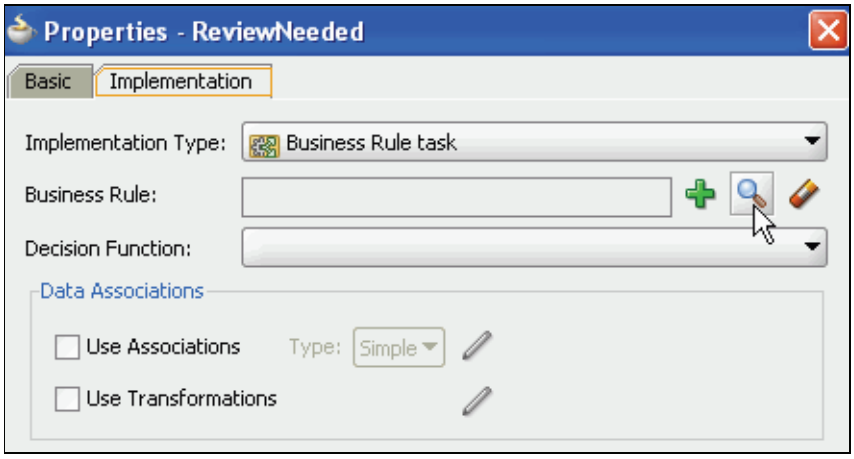
Back in the Business Rule properties window, click **OK**.



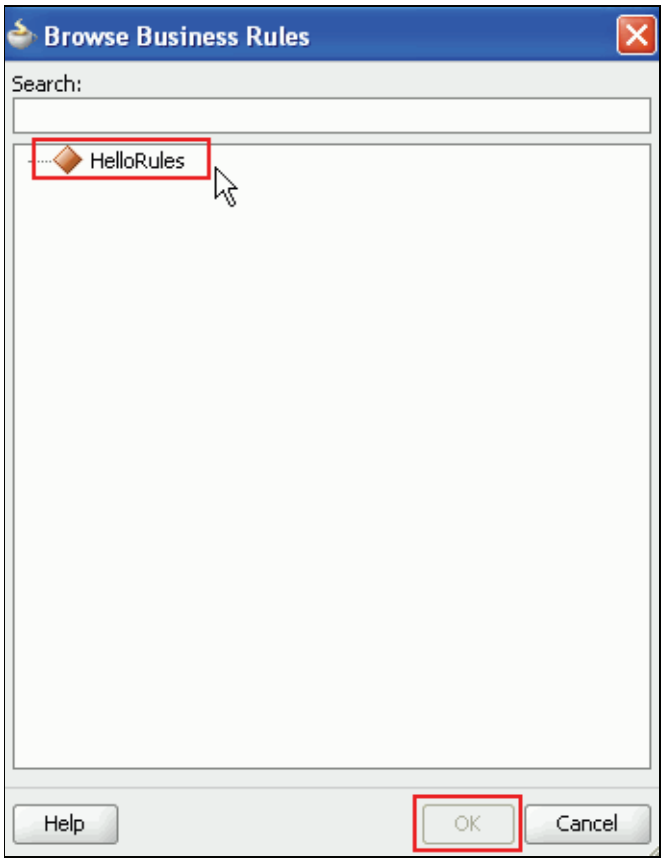
Click **Save All**.

5. You now need to wire the implementation that you just created (the **HelloRules** business rule) to the **ReviewNeeded** business rule activity in the BPM process. Click the **HelloWorldProcess** tab to bring the focus back to the BPM Design Editor.

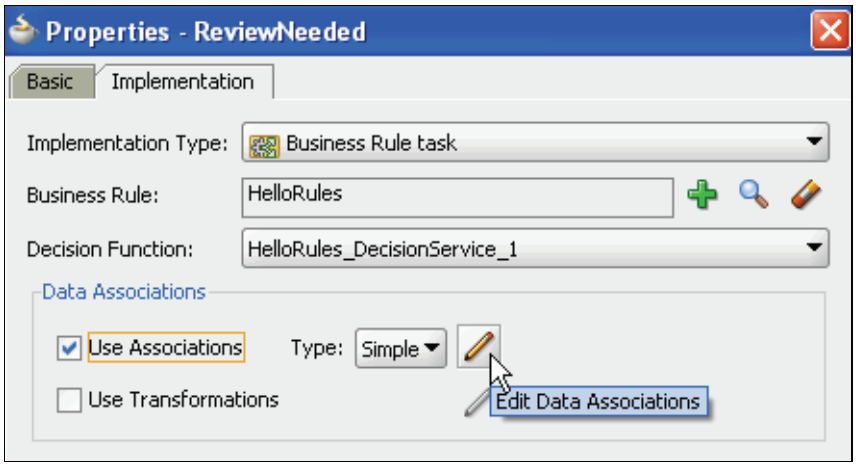
   Right click the **ReviewNeeded** activity to open its properties window. Click the browse button to the right of the **Business Rule** field.



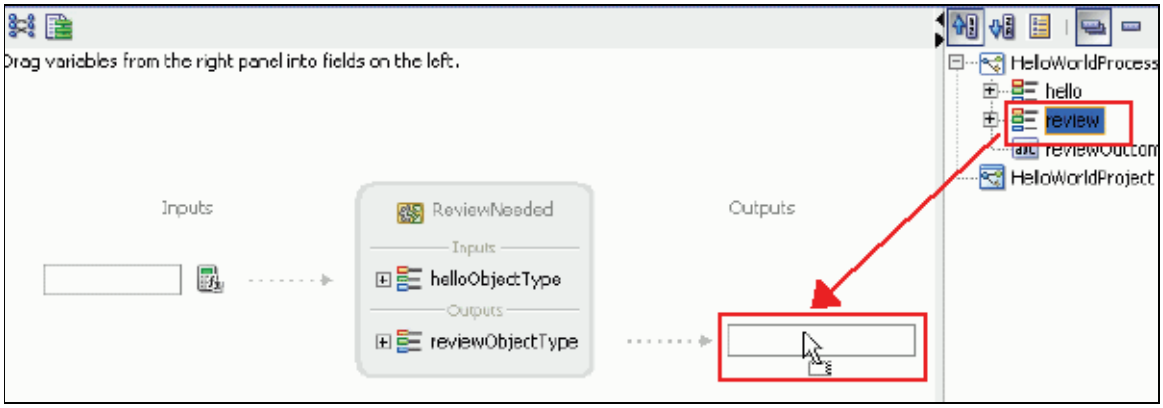
   When the Browse Business Rules window opens, you should now see your **HelloRules** business rule implementation. Select it and click **OK**.



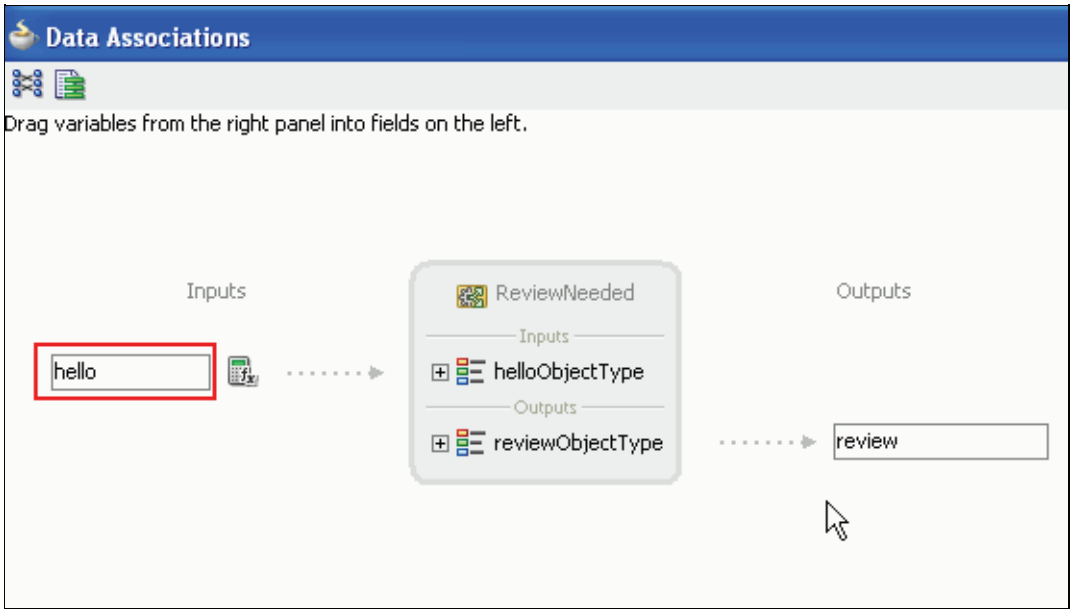
6. In the Properties window, click the checkbox next to **Use Associations**. Keep the default setting for Type, which is Simple associations and click the **Edit Data Associations** button (the pencil icon).



The Data Associations window opens. In this window, you map process data objects (**`hello`** and **`review`**) to the Inputs and Outputs that you defined for the **HelloRules** business rule implementation. From the right column, click and drag the **`review`** data object, dropping it into the **Outputs** field as shown below.



Now click and drag the **`hello`** data object from the right column, dropping it on the Inputs field. Click **OK** to save this mapping and close the Data Associations window.



Click **OK** in the Properties window. Click **Save All**.

Defining a Decision Table for the Business Rule

In this section you define a set of rules that will be applied when the user enters a hello message and greeting. The following table illustrates what those rules are and how they will be applied.

IF ...	**Length is ....**			
greeting:	Short	Medium	Medium	Long
message:	(anything)	Short	Medium or Long	(anything)
ReviewObject:	↓	↓	↓	↓
reason:	"Greeting is too short"	"Length is too short"	null	null
review:	"true"	"true"	"false"	"false"

The lengths of the **`HelloObject.greeting`** and **`HelloObject.message`** strings will be evaluated. You will define four rules that determine the combination of greeting and message lengths and then set the appropriate values to the ReviewObject properties: **`reason`** and **`review`**. For example, if the greeting length is Medium and message length is Short, the ReviewObject's reason property would be set to "Length is too short" and its review property would be set to "true". Consequently, the process would flow to a ReviewNeeded activity and could be accepted or rejected by the reviewer.

A decision table can be designed in many different ways. In our scenario, the decision table consists of four elements as described here:

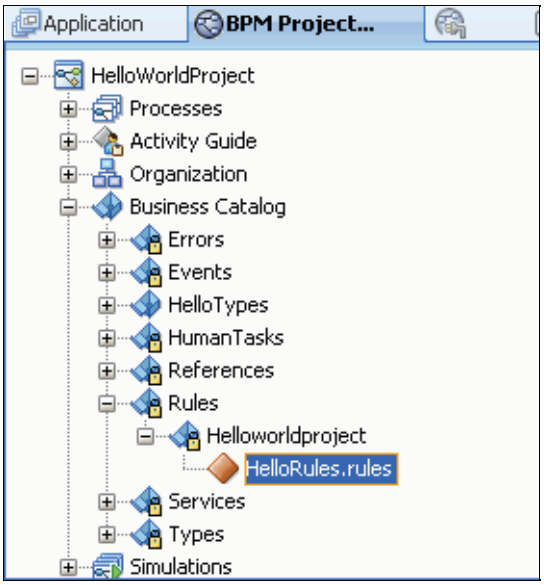
A set of **Conditions** - In this case, each condition is the length of either the *`greeting`* or the *`message`* string variable
A **Range of Values**, called a **bucketset** - For example a variable might be **< 5** characters in length, or **between 5 and 50** characters in length, and so on. These are ranges within a bucketset.
An **Action** - This is the act that should occur when a rule is evaluated. For example, the *`review`* property might be set to "true".
A **Rule** - This is a specific mapping of **condition** > **range** > **action**.

The Decision table is created in five high level steps:

1. Define the conditions (Step 2 below)
2. Define a bucketset (Step 3)
3. Assign that bucketset to each condition (Step 4)
4. Define the action (Step 5)
5. Define the rules (Steps 6 - 9)

1. Open the Rules Editor .

In the BPM Navigator, expand **Business Catalog > Rules > Helloworldproject**. Double click on **HelloRules.rules**.



The Rules Editor opens and, by default, the Ruleset_1 node in the left panel of the editor is selected.

2. Create a decision table for the rule.

Click the green plus sign button on the toolbar and select **Create Decision Table** from the drop-down menu.

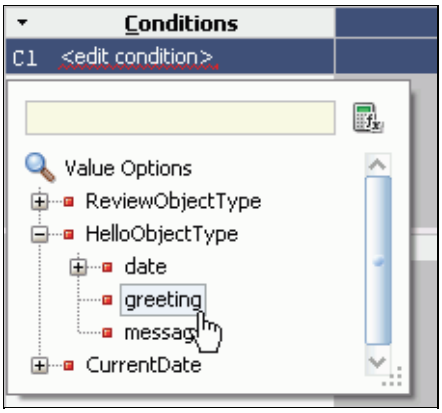
The Ruleset_1 editor changes to provide a table for you to define the decision table. Change the default name of the decision table from **DecisionTable_1** to **`ReviewDecisionTable`**. Click on the field containing the default name. A text field appears below it. Enter the new name and hit Enter to accept the value.

Add two conditions to be checked for the table:

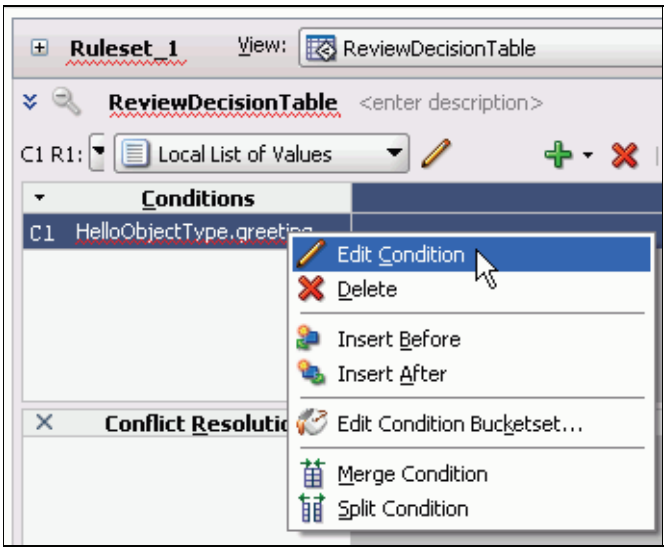
HelloObject.greeting.length( )
HelloObject.message.length( )

Add the first condition by clicking on the **<insert_condition>** field in the table as shown below. This will automatically add a condition row, with a link that says **<edit_ condition>**.

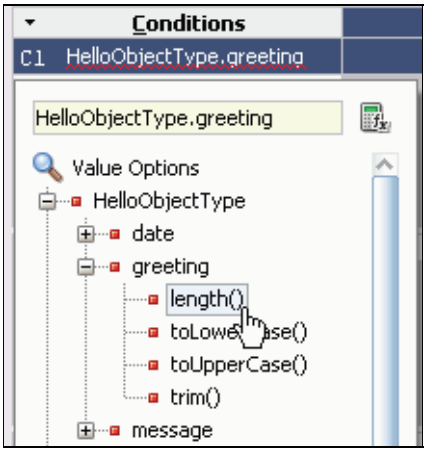
Right click on the **<edit_condition>** field and select **Edit Condition**. A drop-down list appears, displaying the defined data object types within the project. Expand **HelloObjectType** and select **greeting**.



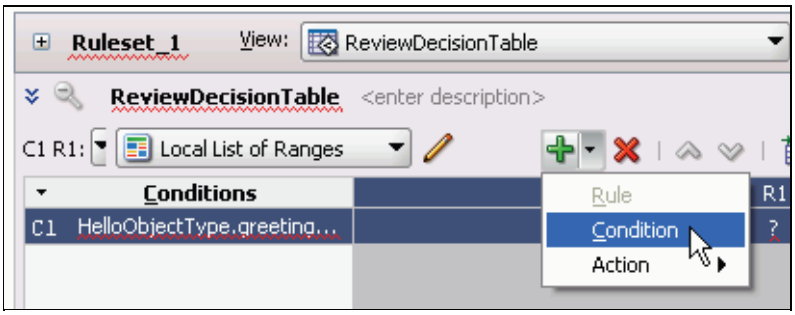
Right click on **HelloObjectType.greeting** and select **Edit Condition** again.



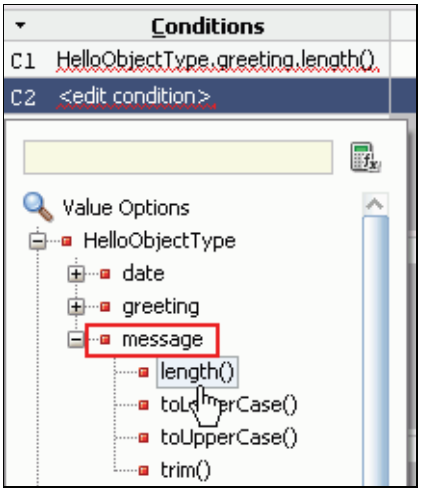
This time, the drop-down list that appears presents more options. Expand **HelloObjectType > greeting** and select the **length( )** function.



Add the second condition by clicking the green plus sign button directly above the Conditions table and selecting **Condition** from the drop-down menu.

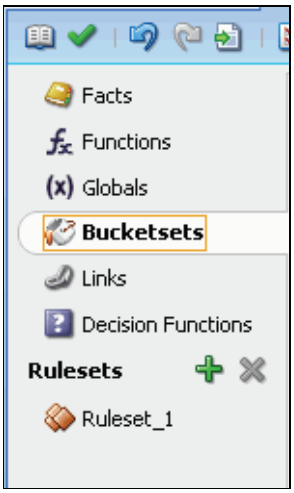


Right click on the newly added condition as you did for the first condition and select **Edit Condition**. This time, you can go directly to adding the length( ) function as part of the condition. Expand **HelloObjectType > message** and select **length( )**.

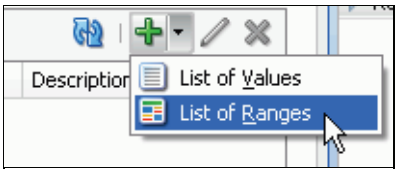


Click **Save All**.

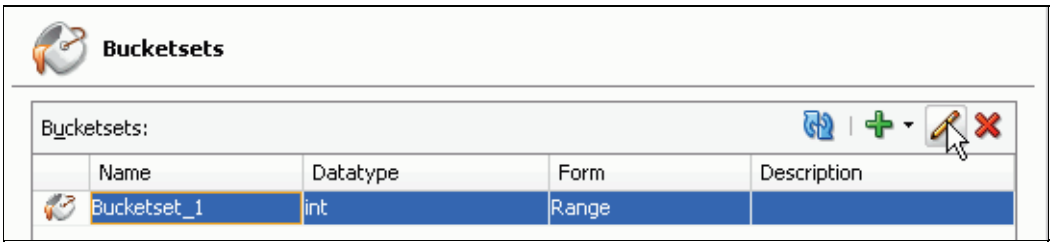
3. The rule will compare each condition against a range of values, called a "bucketset". Click the **Bucketset** node in the left panel of the Rules editor to create the bucketset.



Click the green plus sign button and select List of Ranges from the drop-down menu.

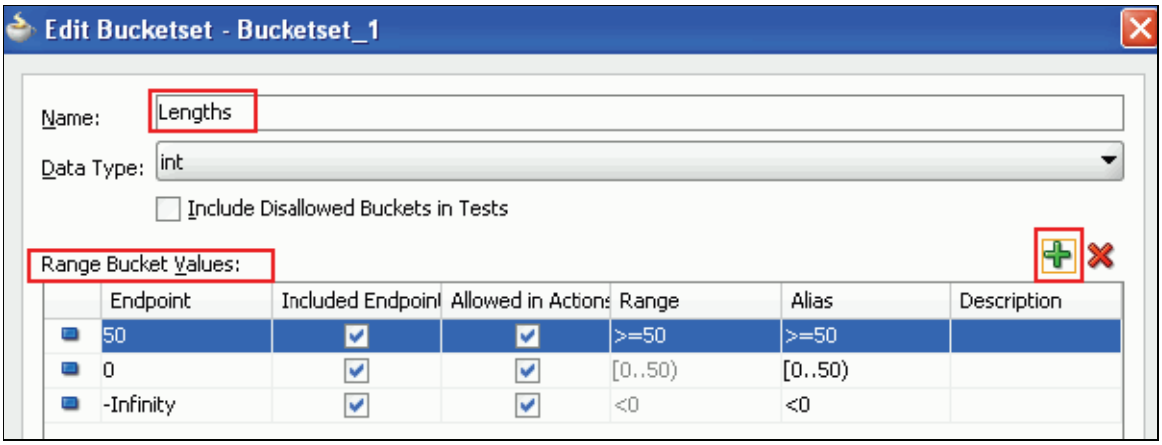


A new bucketset row is inserted. With this row selected, click the pencil icon to edit the list of ranges.



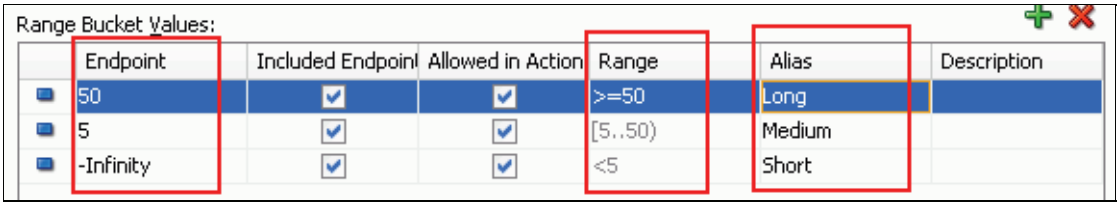
In the Edit Bucketset window that appears, change the value of the **Name** field to `Lengths`. Leave the Data Type field set to int.

The Range Bucket Values table has an initial default row in which the **Endpoint** is set to **-Infinity**. Click the green plus sign button twice more to add two more rows.



Change the `0` (zero) **Endpoint** value to `5` by clicking on the value and typing the new value. Notice that this changes the Range for this row to `[5..50]` and also changes the Range of the row whose endpoint is **'Infinity** to **<5**.

Change the Alias for each row to `Short`, `Medium`, and `Long` as shown below. Double clicking on the default Alias value makes it editable.

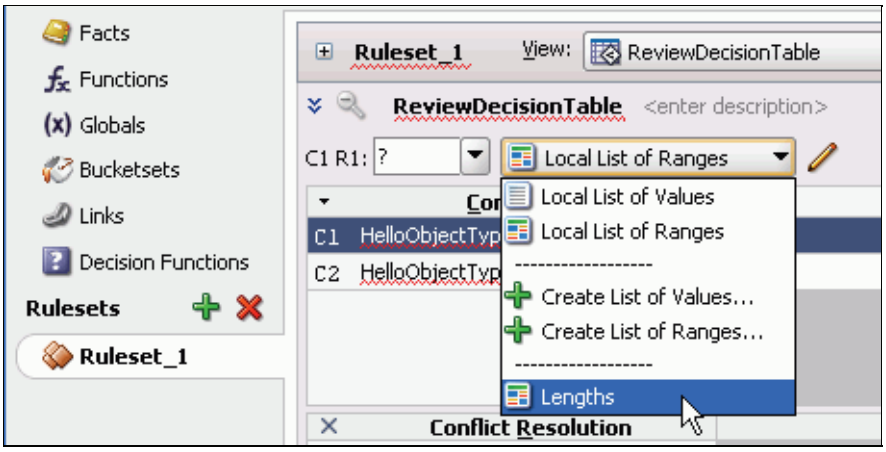


Click **OK** to close the Edit Bucketset window.

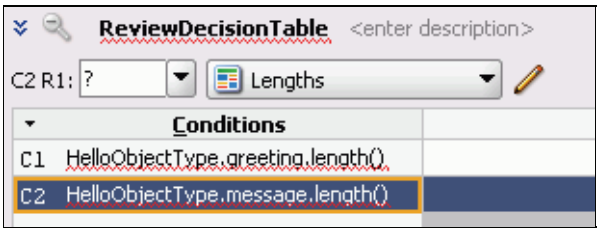
Click **Save**.

4. Assign the Lengths range of values bucketset to each of the conditions in your conditions table. Click the **Ruleset_1** node in the Ruleset editor to bring back the decision table editor.

   Click the **C1** condition to select it and then click the **Local List of Ranges** drop-down list above the Conditions table and select **Lengths**.

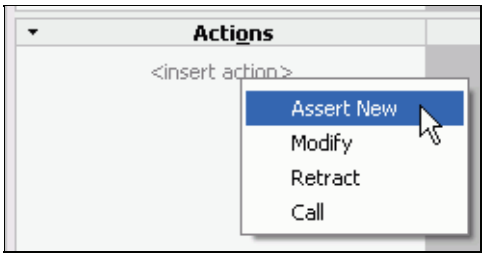


Do the same thing for the **C2** condition.



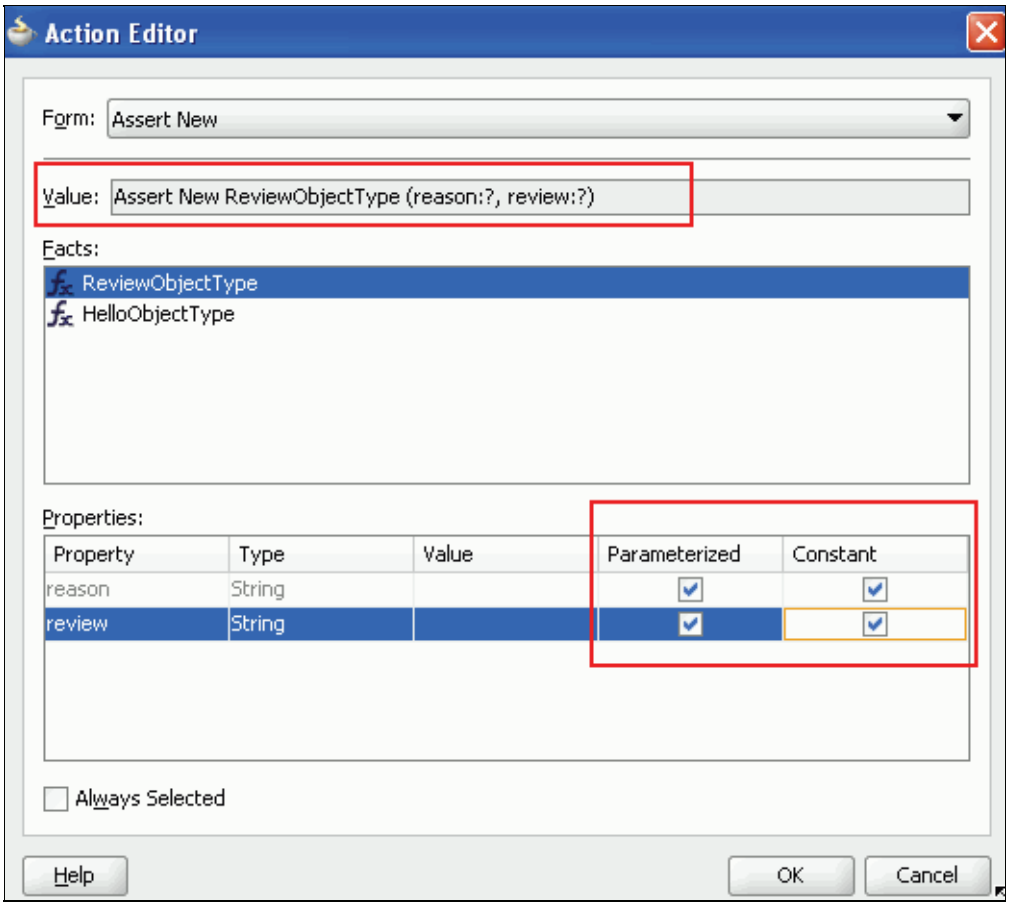
5. You are almost ready to start defining the rules to be applied to each combination of range of values and condition. First you must define an Action to be performed when a rule is implemented.

   Create a new Action. In the **Actions** panel, click the **<insert_action>** field and select **Assert New**.



Once the **assert new ( )** line appears, right click it and select **Edit**.

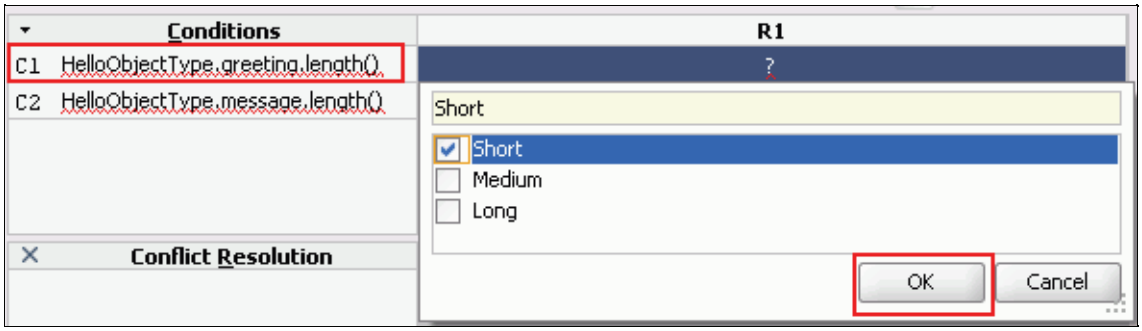
The **Action Editor** window opens. Select **ReviewObjectType** from the Facts panel. The properties of this object type now appear in the Properties panel in the lower portion of the window. Check the **Parmeterized** and **Constant** checkboxes for both the **reason** and the **review** properties.



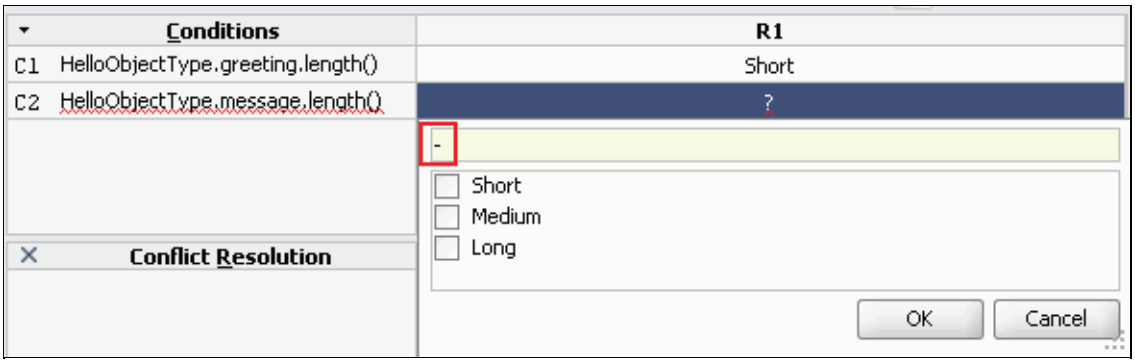
Notice that the effect of this is that an **`assert new( )`** statement is being constructed in the Value field. It instantiates a ReviewObject with values for the **`reason`** property (a String), and the **`review`** property (a String) You can see this assert statement in the screenshot above. Each rule that you define in the next step will provide specific values for these parameters.

Click **OK**, then **Save**.

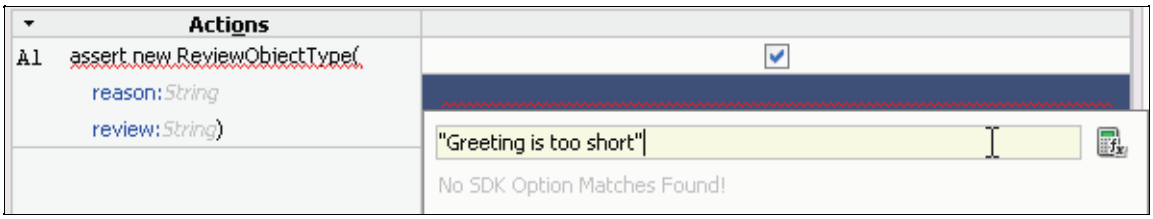
6. Define the first rule. Click on the **C1** row beneath the **R1** column. A drop-down list appears representing the bucketset you defined for this ruleset. Check **Short** and click **OK**.



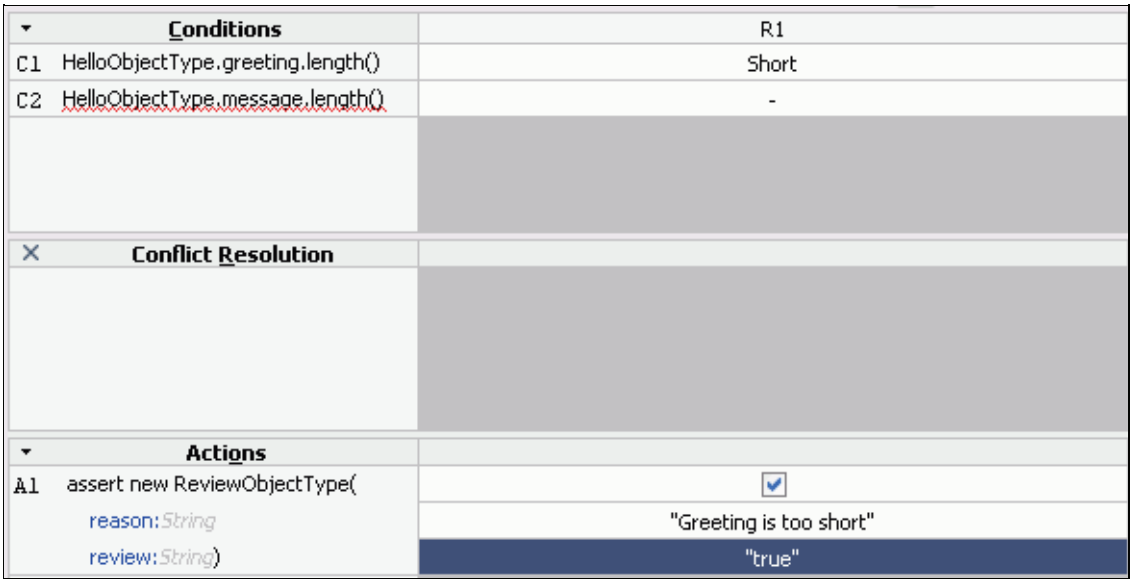
Click the **R1** column for the **C2** row. This is the length for the **`message`** property. In this case, it doesn't matter what the length is, so enter a dash (-) and click **OK**.



In the **Actions** panel, click the checkbox in the column to the right of the **`assert new`** statement, indicating that you *do* want the ReviewObject asserted for this rule. Then click the row beneath the checkbox, next to the **`reason`** property. When the text field appears, type: **`"Greeting is too short"`**. Hit Enter when finished.

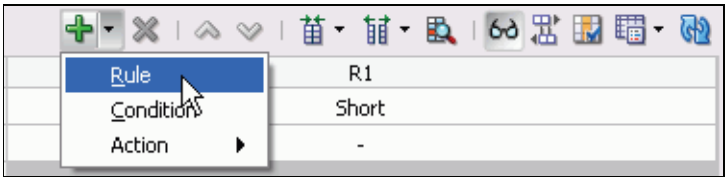


Click the column next to the review property and enter the value **`"true"`** into the text field. Hit Enter when finished.



You have just defined **Rule 1**, indicating that if the *`greeting`* is "Short" (<5 characters), the message is subject to review (*`review`* = "true") and the *`ReviewObject.reason`* should be set to "Greeting is too short".

7. Define the second rule. Click the green plus sign button above the Rules column and select **Rule**.



Notice that when the new rule column is added, it is added with the heading **R1** and the rule you just defined is moved over to the right, under the heading **R2**.

ReviewDecisionTable <enter description>		
A1 R1:		
Conditions	R1	R2
C1 HelloObjectType.greeting.length()	?	Short
C2 HelloObjectType.message.length()	?	-
Conflict Resolution		
Actions		
A1 assert new ReviewObjectType(	☐	☑
reason: String		"Greeting is too short"
review: String)		"true"

Beware: When you make the first edit to the new rule, it will switch back to its original location and original heading name. **Always be sure that you are editing the rule you intended to edit.**

Make edits to this rule, just as you did the previous rule, using the following table to guide you:

Section of Decision Table	Property	Value to Set
Condition	HelloObject.*greeting*.length( )	Medium
Condition	HelloObject.*message*.length( )	Short
Action	ReviewObject.*reason*	`"Length is too short"`
Action	ReviewObject.*review*	`"true"`

Conditions	R1	R2
C1 HelloObjectType.greeting.length()	Short	Medium
C2 HelloObjectType.message.length()	-	Short
Conflict Resolution		
Actions		
A1 assert new ReviewObjectType(	☑	☑
reason: String	"Greeting is too short"	"Length is too short"
review: String)	"true"	"true"

8. Define the last rule that appears in the table as shown at the beginning of this section. Define it in the same way you defined the rule in the preceding step. The table below provides the values you will use.

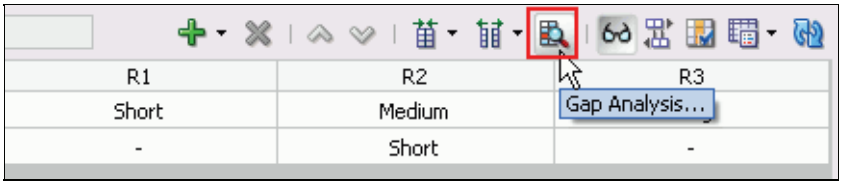
Section of Decision Table	Property	Value to Set

Condition	HelloObject.*greeting*.length( )	Long
Condition	HelloObject.*message*.length( )	–
Action	ReviewObject.*reason*	`null`
Action	ReviewObject.*review*	`"false"`

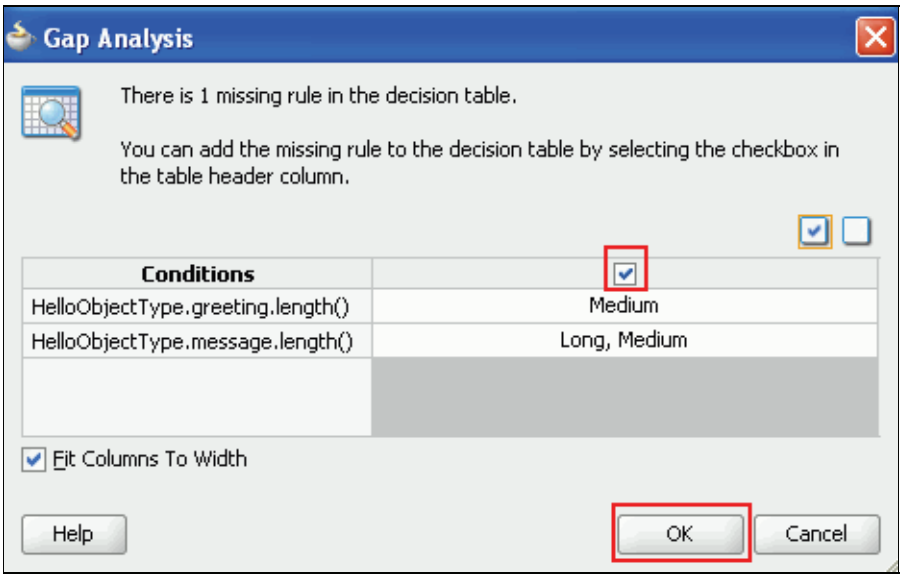
Note that you must select the `null` from the drop-down list, rather than type it into the text field.

Conditions	R1	R2	R3
C1 HelloObjectType.greeting.length()	Short	Medium	Long
C2 HelloObjectType.message.length()	-	Short	-
Conflict Resolution			
Actions			
A1 assert new ReviewObjectType(	☑	☑	☑
reason: String	"Greeting is too short"	"Length is too short"	null
review: String)	"true"	"true"	"false"

9. Run the **Gap Analysis** tool to automatically create the final rule. It will determine which conditions have not been covered by the existing rules. Click the Gap Analysis button on the Decision Table editor toolbar.



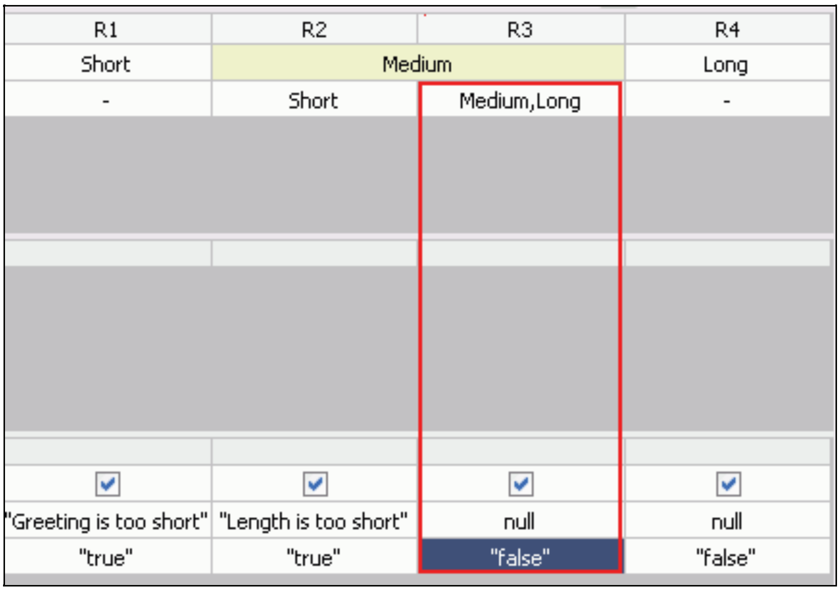
The Gap Analysis window opens showing the rule that it determined was missing. Click the checkbox above the rule to allow the rule to be included in the decision table. Click **OK**.



The rule now appears in the decision table as Rule 3 and the rule that you defined in the last step moves to the

Rule 4 position. Assert the action for Rule 3 (Column **R3**) and add the following action values:

```
reason = null
review = "false"
```



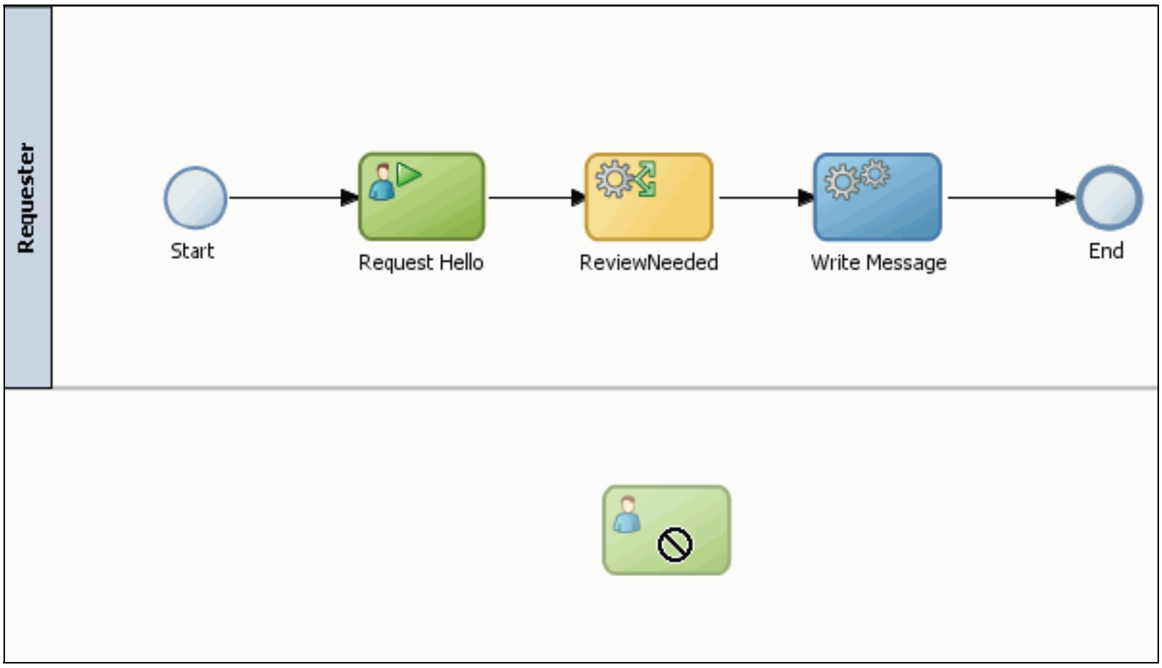
R1	R2	R3	R4
Short	Medium		Long
-	Short	Medium,Long	-
☑	☑	☑	☑
"Greeting is too short"	"Length is too short"	null	null
"true"	"true"	"false"	"false"

Click **Save** and close the **HelloRules.Rules** tab.

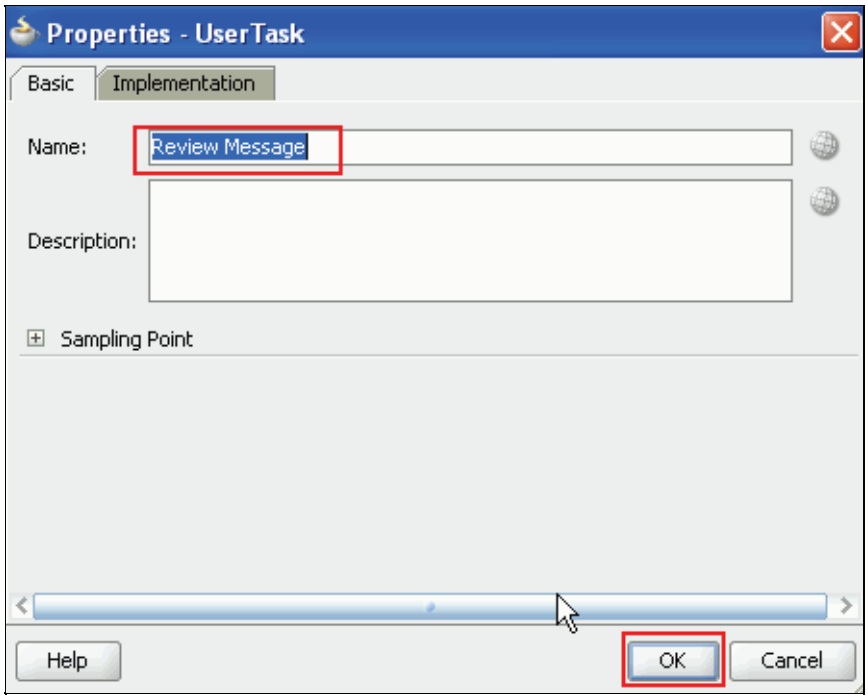
Adding the Review Message Human Interaction

1. Add a new human interaction activity to the HelloWorldProcess. If necessary, click the HelloWorldProcess tab in the Design Editor to bring it to the front.

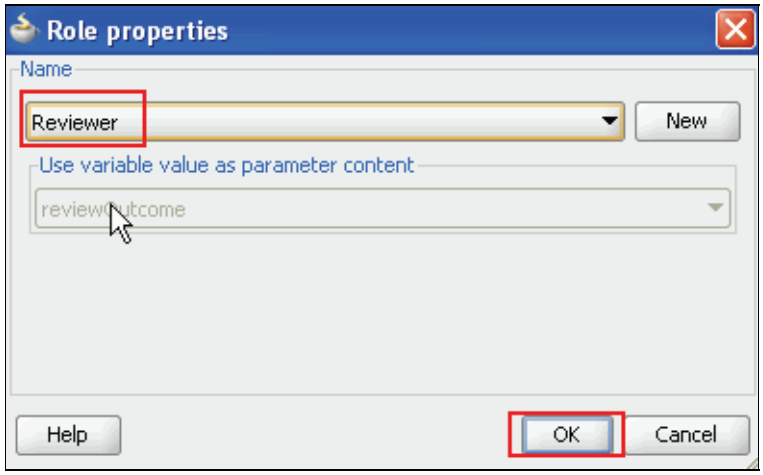
   Expand the Activities accordion panel in the Component Palette and, from the Interactive section, click and drag a User activity, dropping it *outside* of the Requester lane and directly below the ReviewNeeded activity.



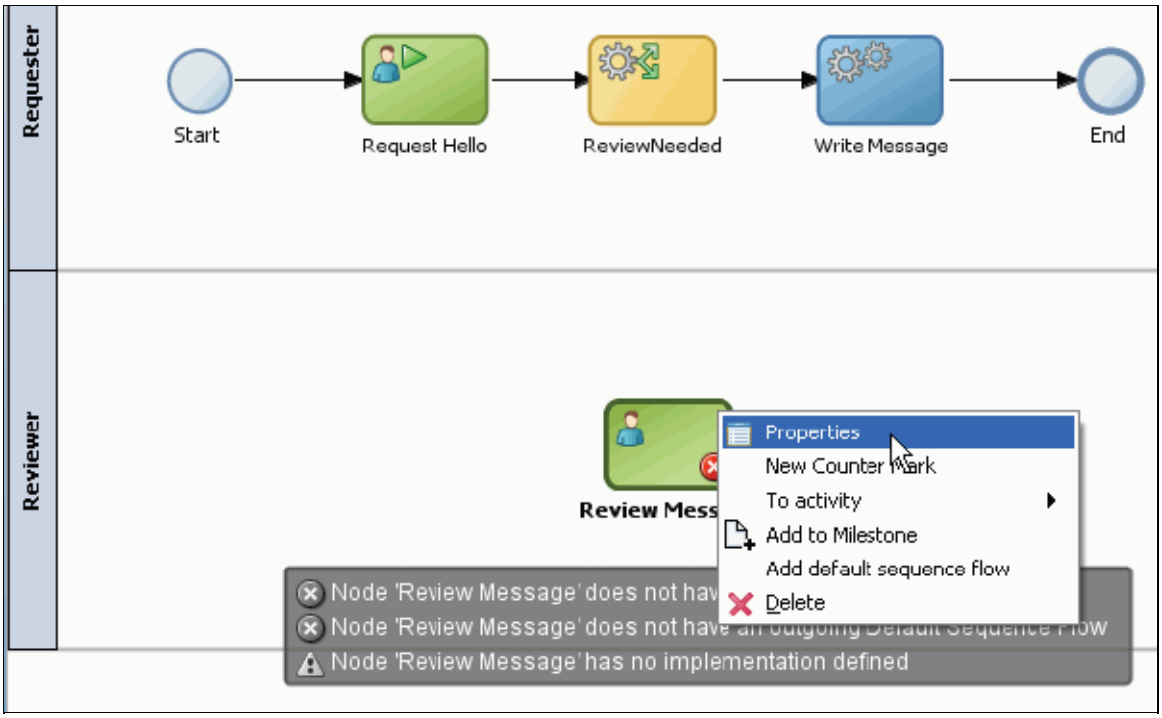
When the Properties window opens, name the activity `Review Message` and click **OK**.



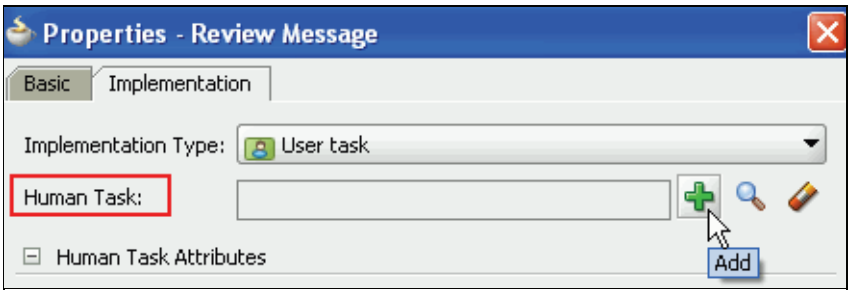
You will then be prompted to assign a role to this activity. Select the **Reviewer** role from the drop-down list. Click **OK**.



2. Define the implementation for the Review Message activity. In the Design Editor, right click **Review Message** and select **Properties**.

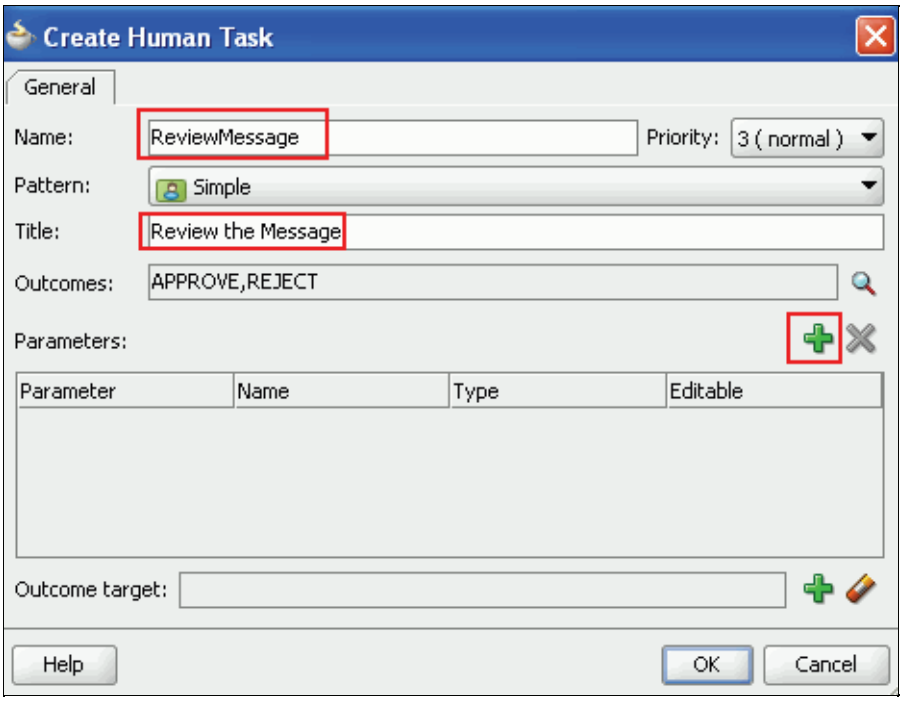


In the Properties window, click the green plus sign button next to the **Human Task** field to define a new human task implementation.



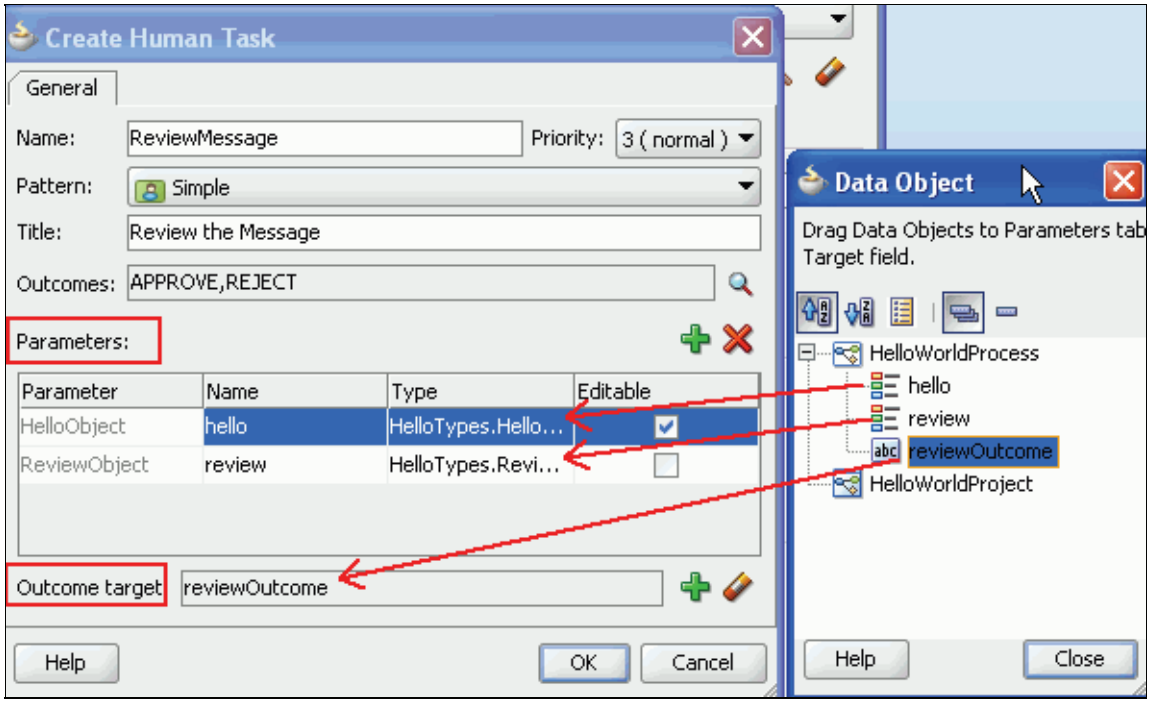
In the Create Human Task window, define the following properties. Accept the other default values.

Name: **`ReviewMessage`**
Title: **`Review the Message`**



Click the **green plus sign button** next to the **Parameters** panel as shown above to add parameters.

The **Data Object** popup window appears next to the Create Human Task window. Drag the `hello` and `review` process data objects into the **Parameters** panel. Select the **Editable** checkbox for the **hello** data object only. Also drag the `reviewOutcome` process data object into the **Outcome Target** field.



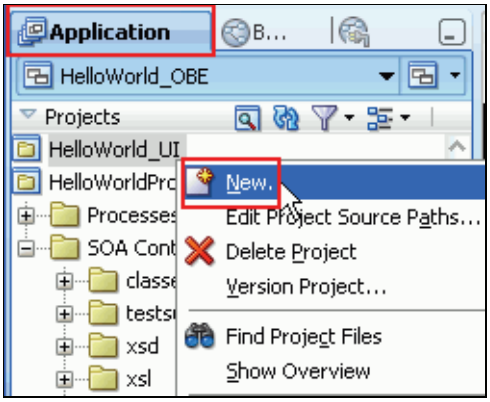
Click **Close** in the Data Object popup window and click **OK** in the Create Human Task window.

Click **OK** in the Properties window.

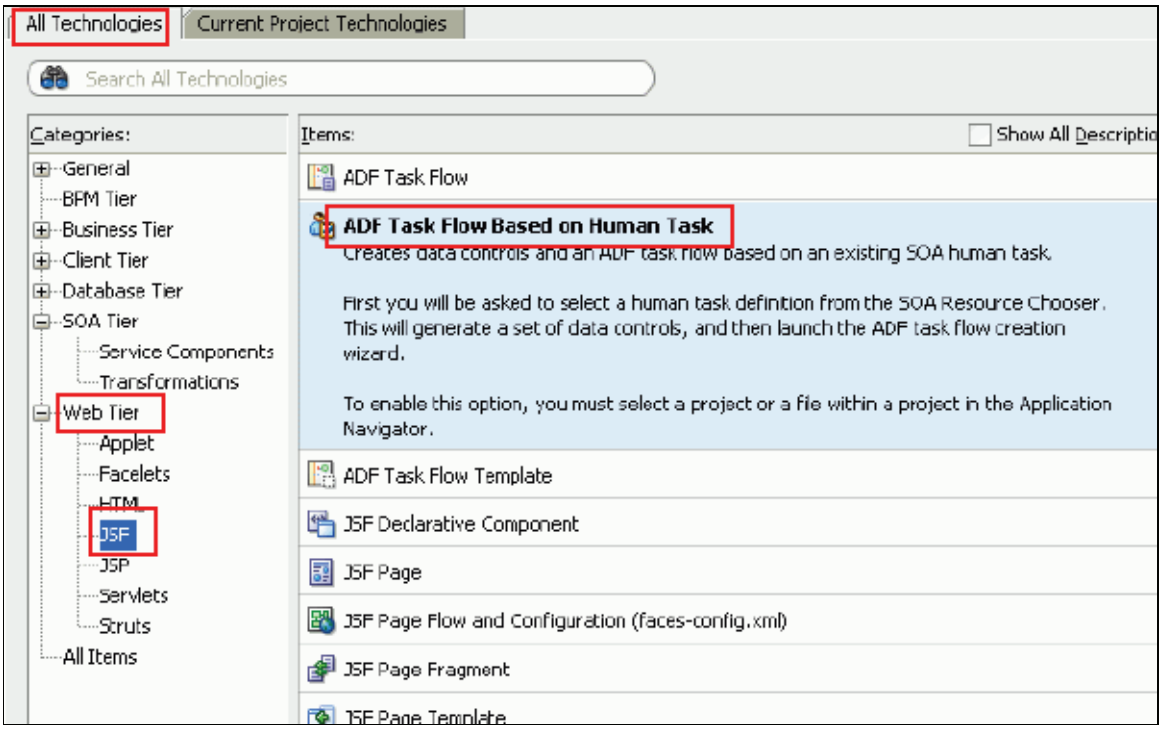
Click **Save All**.

3. Create a new task flow, based on the ReviewMessage human task that you created in the previous step.

Click on the **Application Navigator** tab and then right click on the **HelloWorld_UI** project node and select **New...**



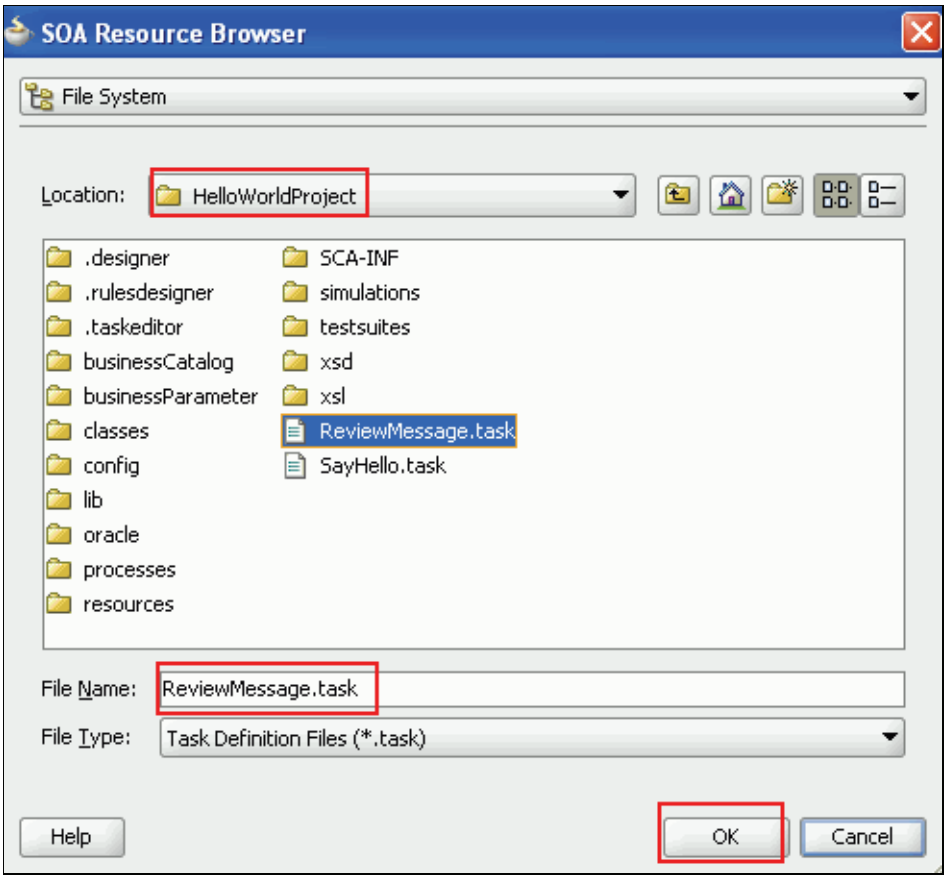
The New Gallery wizard opens. Click the **All Technologies** tab. Expand the **Web Tier** category and select **JSF**. From the Items panel, select **ADF Task Flow Based on Human Task**. By using this approach to creating the taskflow, you can store the taskflow in the same project that is storing your earlier human taskflow, rather than have separate projects for each taskflow.



Click **OK**.

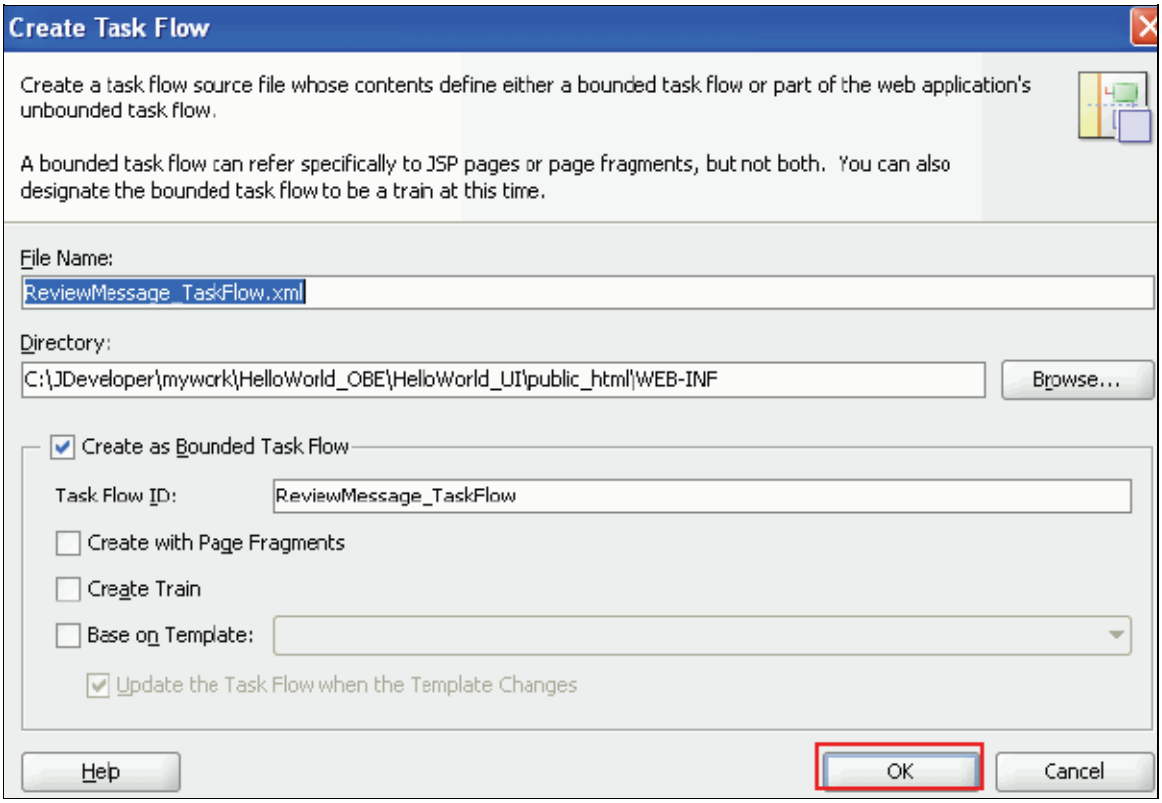
You are now prompted to identify the XML file that was generated when you defined the ReviewMessage human task. The **SOA Resource Browser** window opens.

It is currently looking at the `HelloWorld_UI` folder. Navigate up one level to the `HelloWorld_OBE` folder and then open the `HelloWorldProject` folder. Select the file, `ReviewMessage.task`.



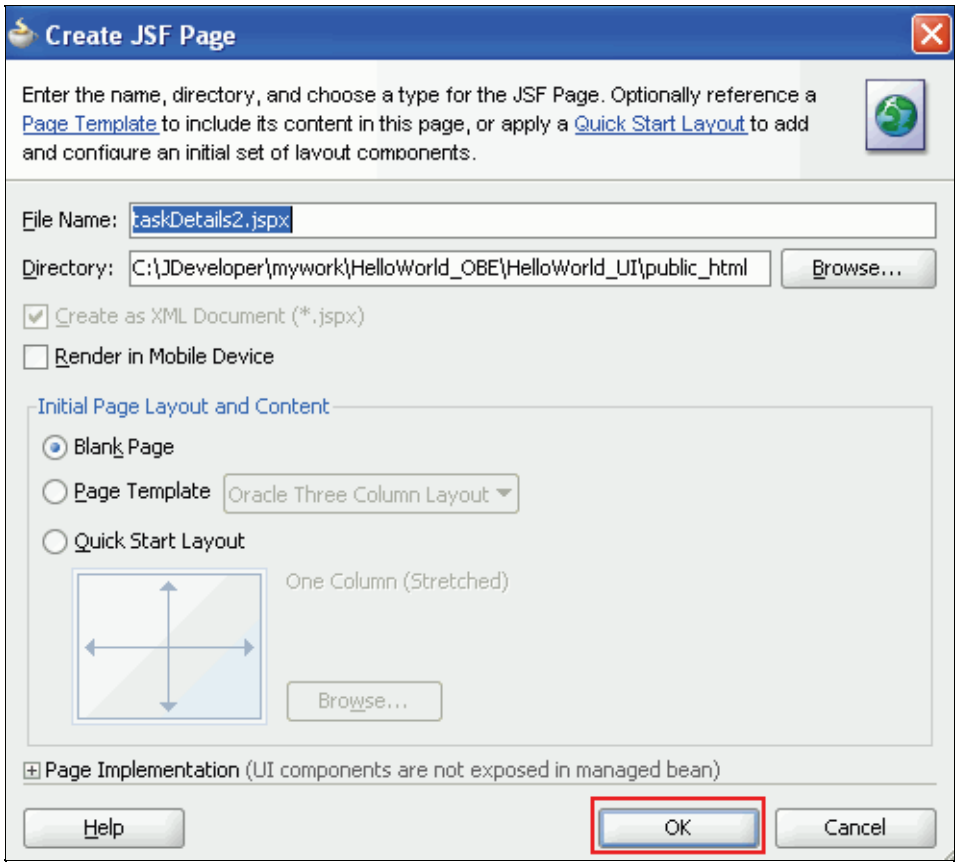
Click **OK**.

In the **Create Task Flow** window, accept all the default values and click **OK**.



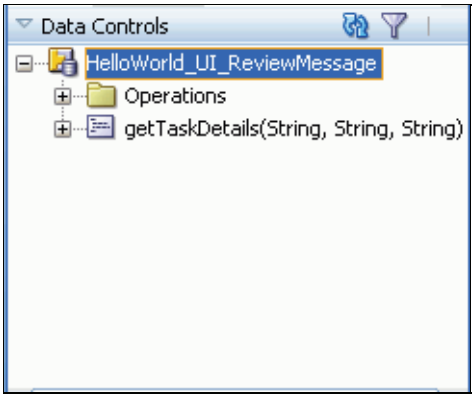
4. The bounded task flow that you just created, using the wizard, now appears in the Task Flow editor (Diagram tab). It does not yet have a web page associated with the **taskDetails2_jspx**. This is the reason that you see an error indication. Recall that in the basic Hello World process, you allowed Studio to automatically generate a form for the Request Hello task flow. For the Review Message task flow, you create your own.

   Double click the **taskDetails2_jspx** icon in the Task Flow editor. The Create JSF Page window opens. Accept all defaults and click **OK**.

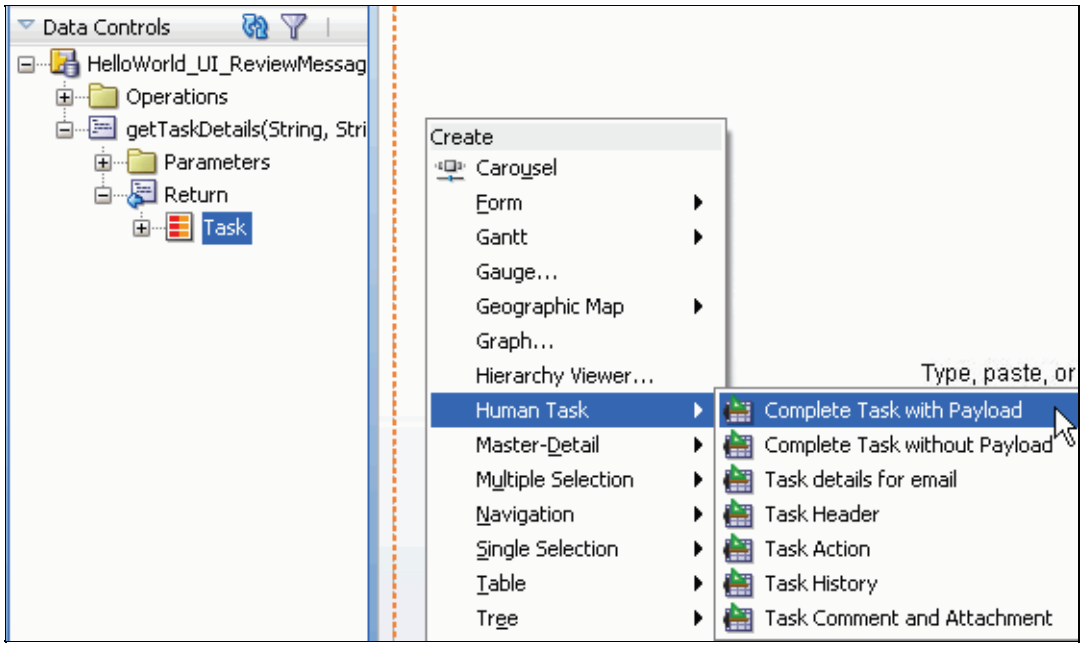


   It will take several seconds for the JSF page designer to become initialized.

5. You should resize some of the accordion panels in the left side of the Studio window now in order to make the **Data Controls** accordion panel larger, since you will be selecting components from it in this step.

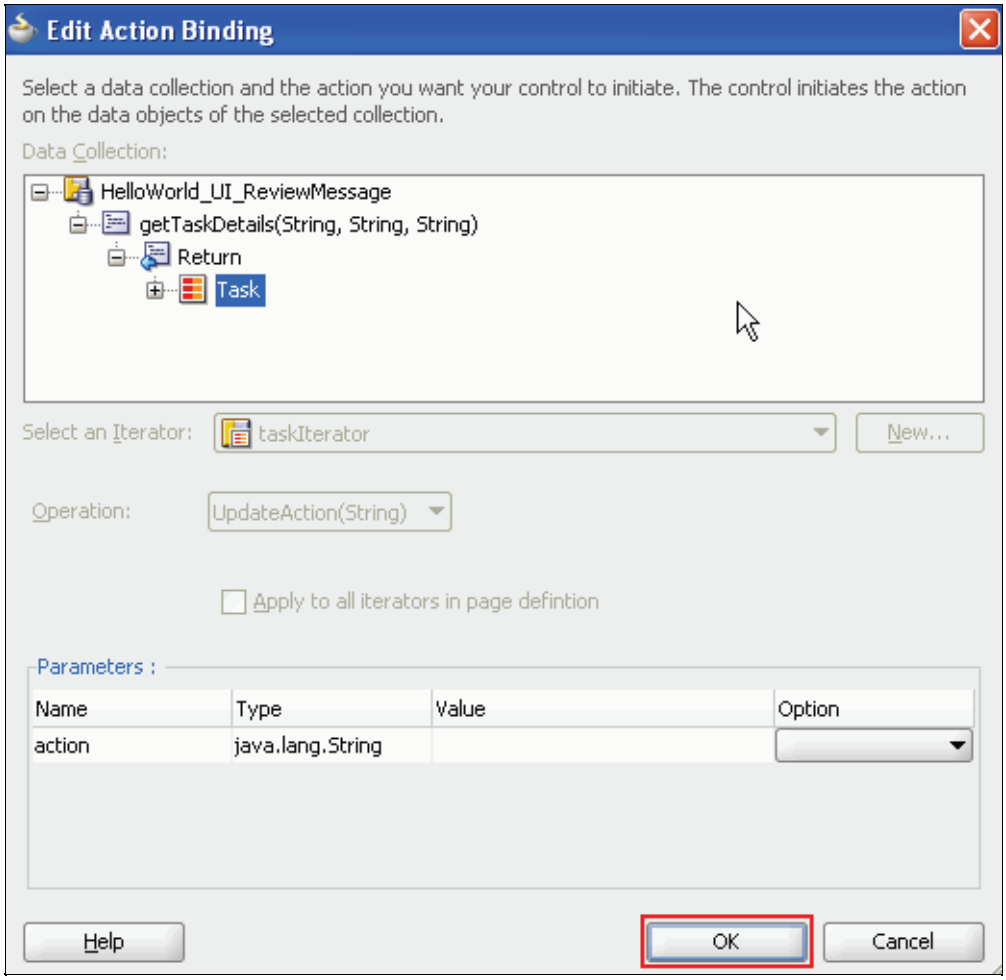


   Expand **HelloWorld_UI_ReviewMessage > getTaskDetails > Return**. Click **Task** and drag it into the Taskflow Design editor. When you drop it, a menu appears. Select **Human Task > Complete Task with Payload**.

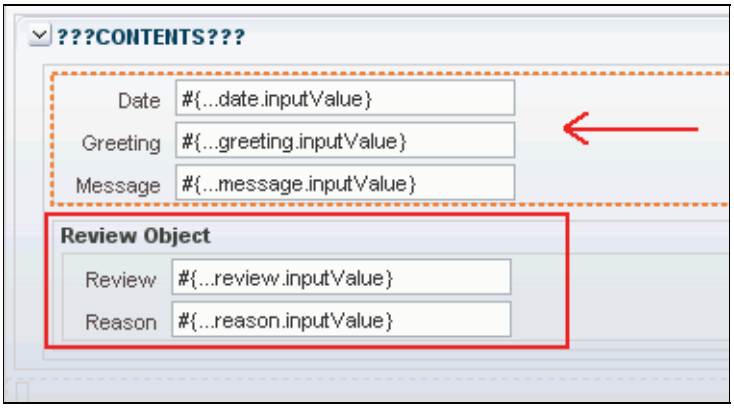


6. The **Edit Action Binding** window opens next. If you are unable to expand the **HelloWorld_UI_ReviewMessage** node, click **OK** in this window.

   A *second* **Edit Action Binding** window will appear. Expand **HelloWorld_UI_ ReviewMessage > getTaskDetails(...) > Return**. Select the **Task** object. Click **OK**.



After a few seconds, the JSF page appears in the design editor (in design mode). Notice that it contains two boxes, the upper one to accommodate the contents of the message entered by the user, and the lower one to accommodate the review comments by the reviewer.



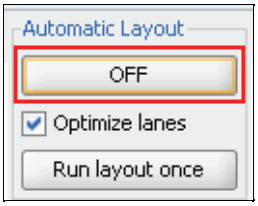
You are now finished with the taskflow and JSF page. Click **Save All**. You can close all tabs except for the HelloWorldProcess tab.

Adding Conditional Branching

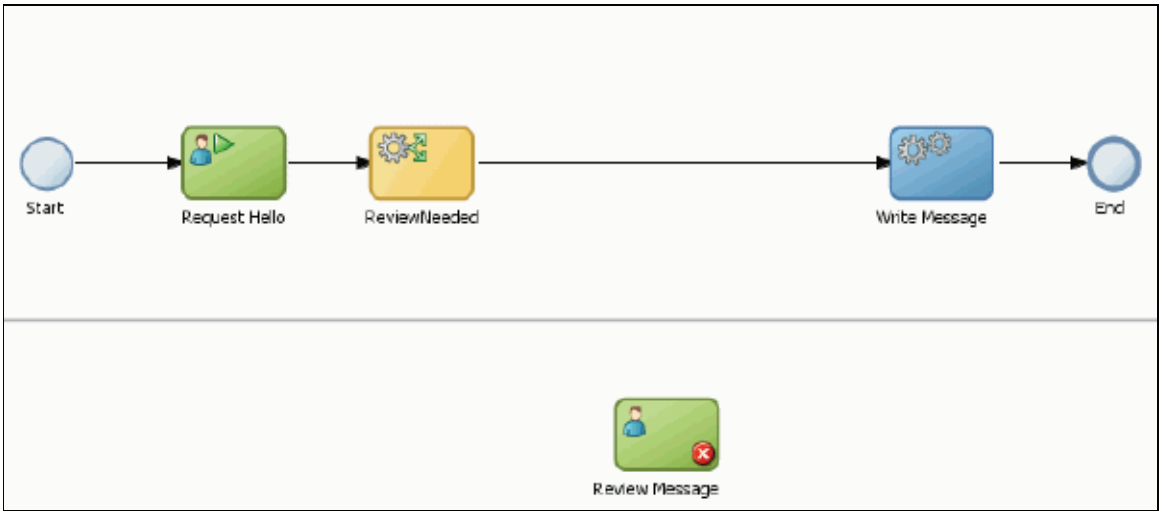
1. Add an Exclusive Gateway to the process model.

   You will be moving design elements around quite a bit in this section and, by default, the Design Editor has Automatic Layout turned ON. You may wish to turn it off so that it does not undo your moves. Click the **Layout** button on the toolbar.

   On the Automatic Layout menu that appears, click the **ON** button, which toggles it to an **OFF** button. Note that you can come back to this menu and get a one time auto-layout at any time or choose to turn automatic layout back on.

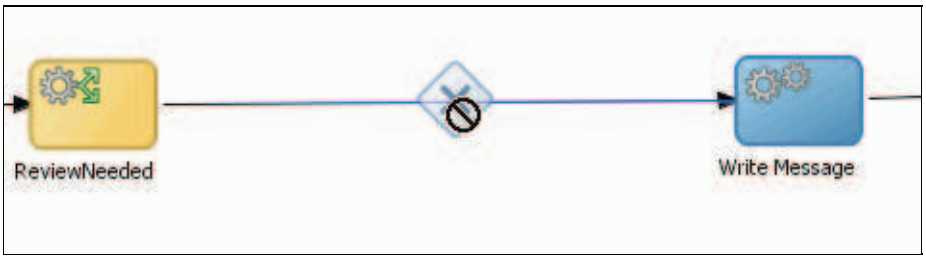


   Make room for changes to the process model by moving the **End** and **Write Message** activities further to the right.

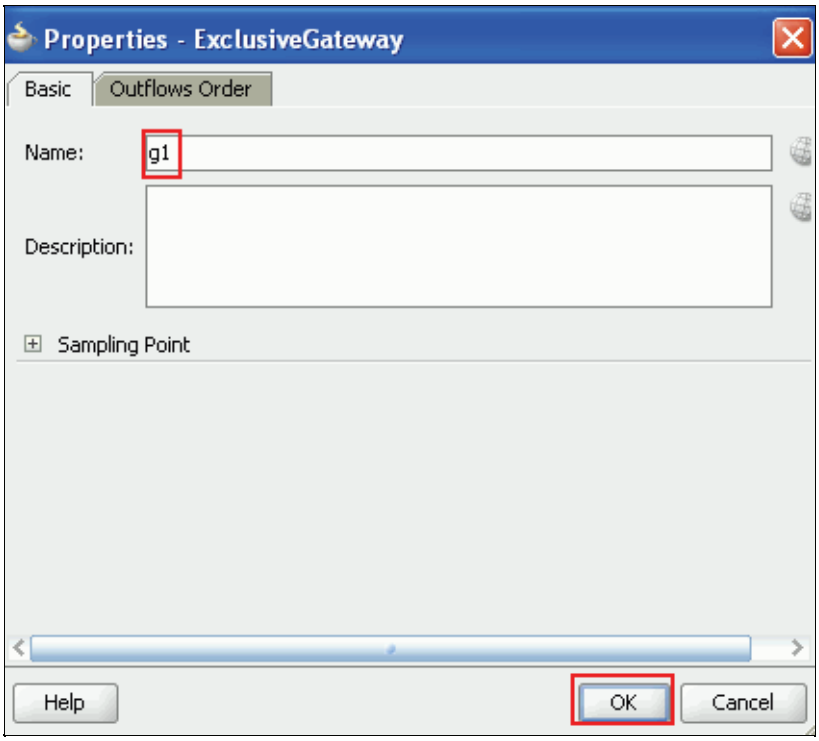


From the Gateway accordion panel of the Component Palette, click and drag an Exclusive Gateway, dropping it on the sequence flow between ReviewNeeded and Write Message.

Important: The line must appear **blue** when you drop the gateway in order for the transition line to connect to it properly. You will probably have to drop it near the center of the space between activities in order to pick up the blue line. You can move the gateway further to the left after you have finished placing it.

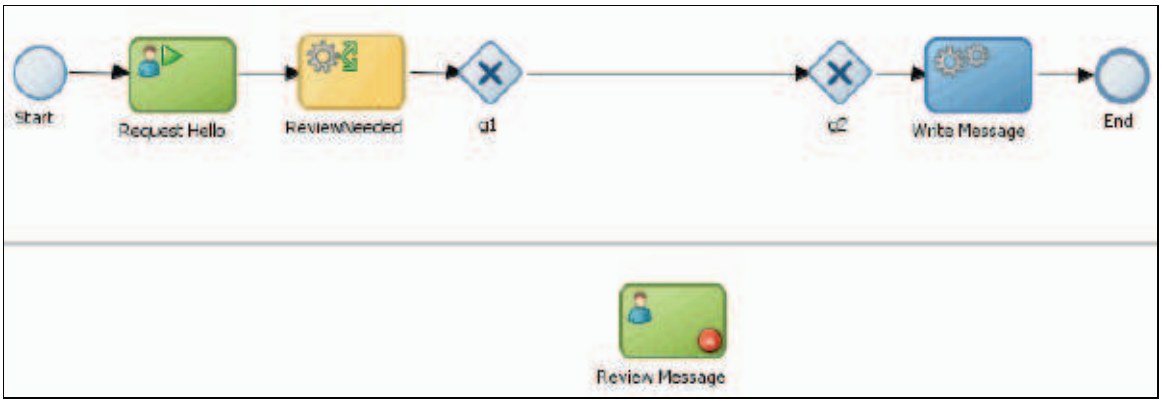


When you drop it, the Properties window opens. Change the name of the gateway to `g1` and click **OK**.

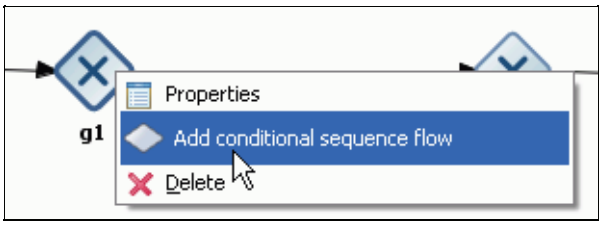


Move the g1 gateway over to the left to make room for another exclusive gateway that you will add in the next step.

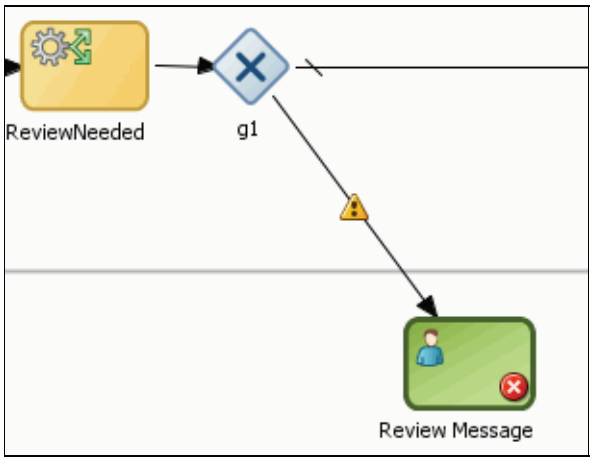
2. Add a second exclusive gateway to the transition line between **g1** and **Write Message**. Follow the same procedure you did in the last step. Name this gateway `g2`.



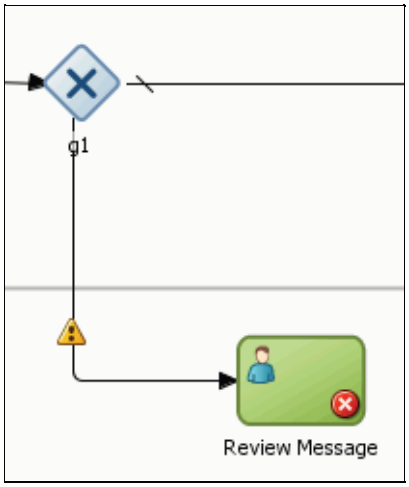
3. Add a conditional sequence flow from **g1** to **Review Message**. Right click on **g1** and select **Add conditional sequence flow**.



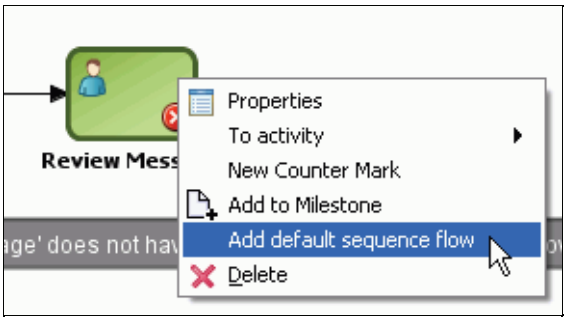
Connect the transition line by clicking on **Review Message**. Do not worry about the warning message regarding lack of default sequence flow. You will fix that in the next step.



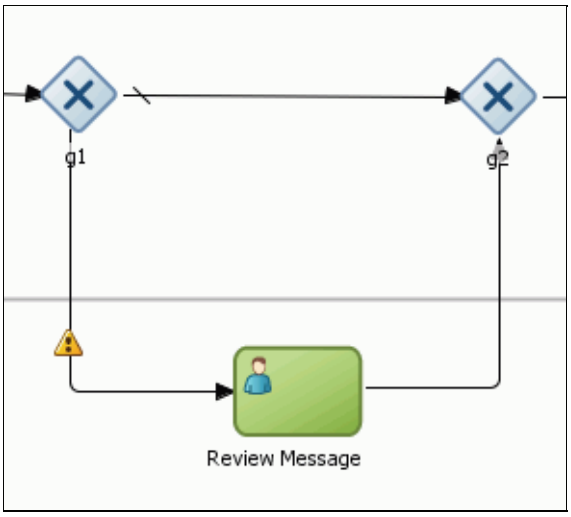
Reshape the transition line as shown below by clicking and dragging it into the desired shape.



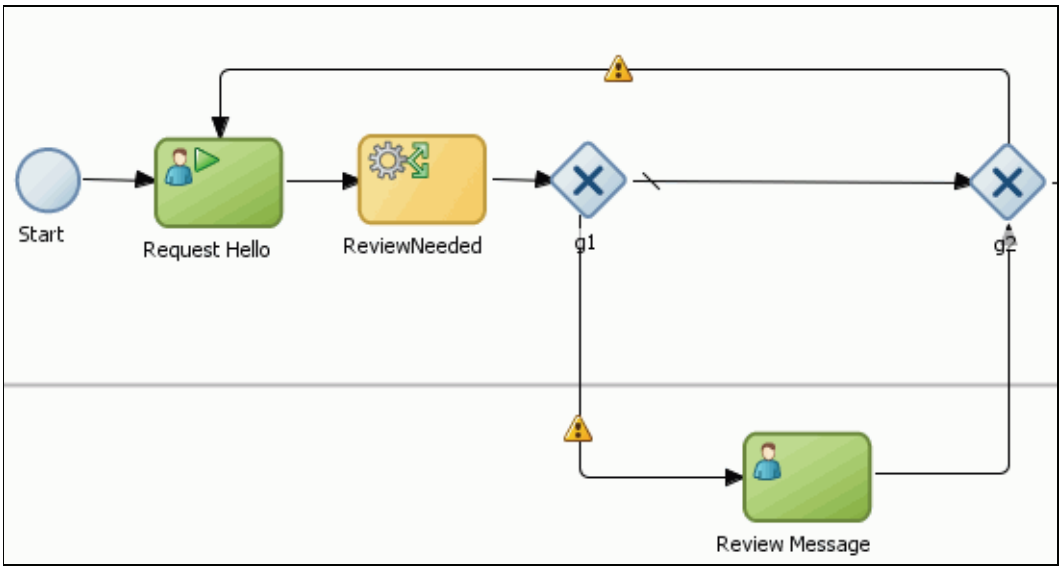
4. Add a default sequence flow from **Review Message** to **g2**. Right click on **Review Message** and select **Add default sequence flow**.



Connect the other end of the sequence flow by clicking on g2. Reshape the transition line as shown below, as you did in the previous step.

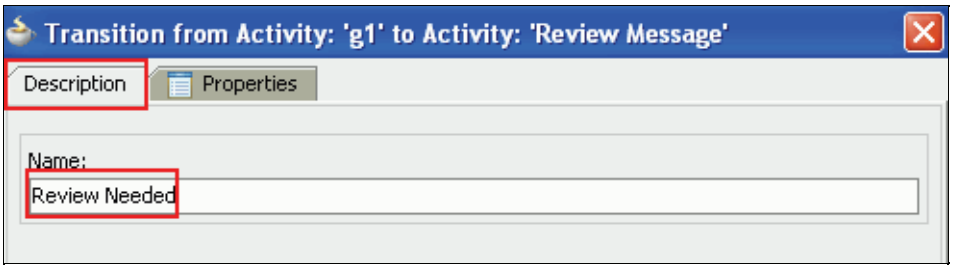


5. Add a conditional sequence flow from **g2** back to **Request Hello**. Reshape the transition line as you have the other two transition lines. (It will be obscured by the existing main sequence flow) It should look like this when you've finished.

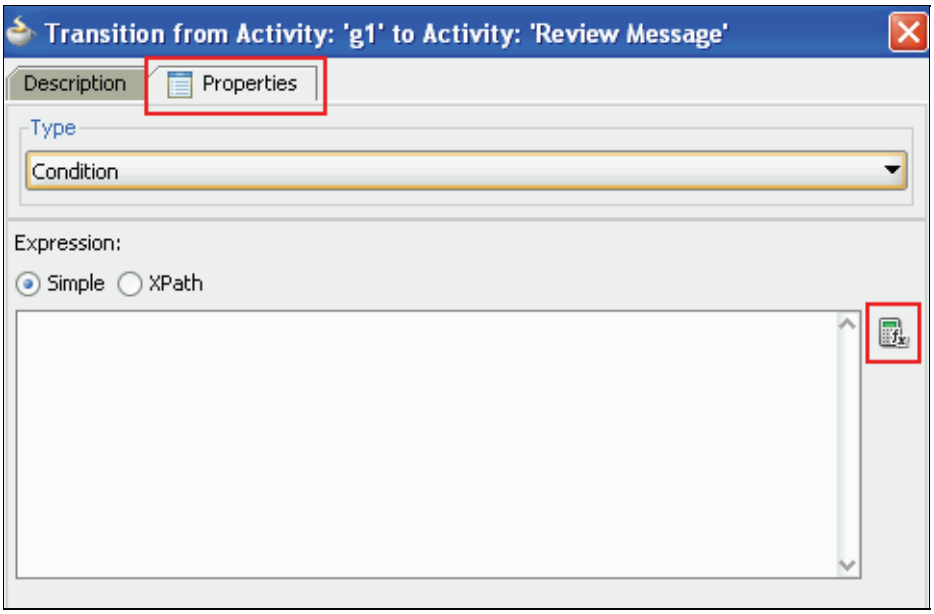


6. Define the condition for the sequence flow from **g1** to **Review Message**. Double click on the transition line between the two objects. The Transition properties window opens.

   On the **Description** tab, name the transition `Review Needed`.



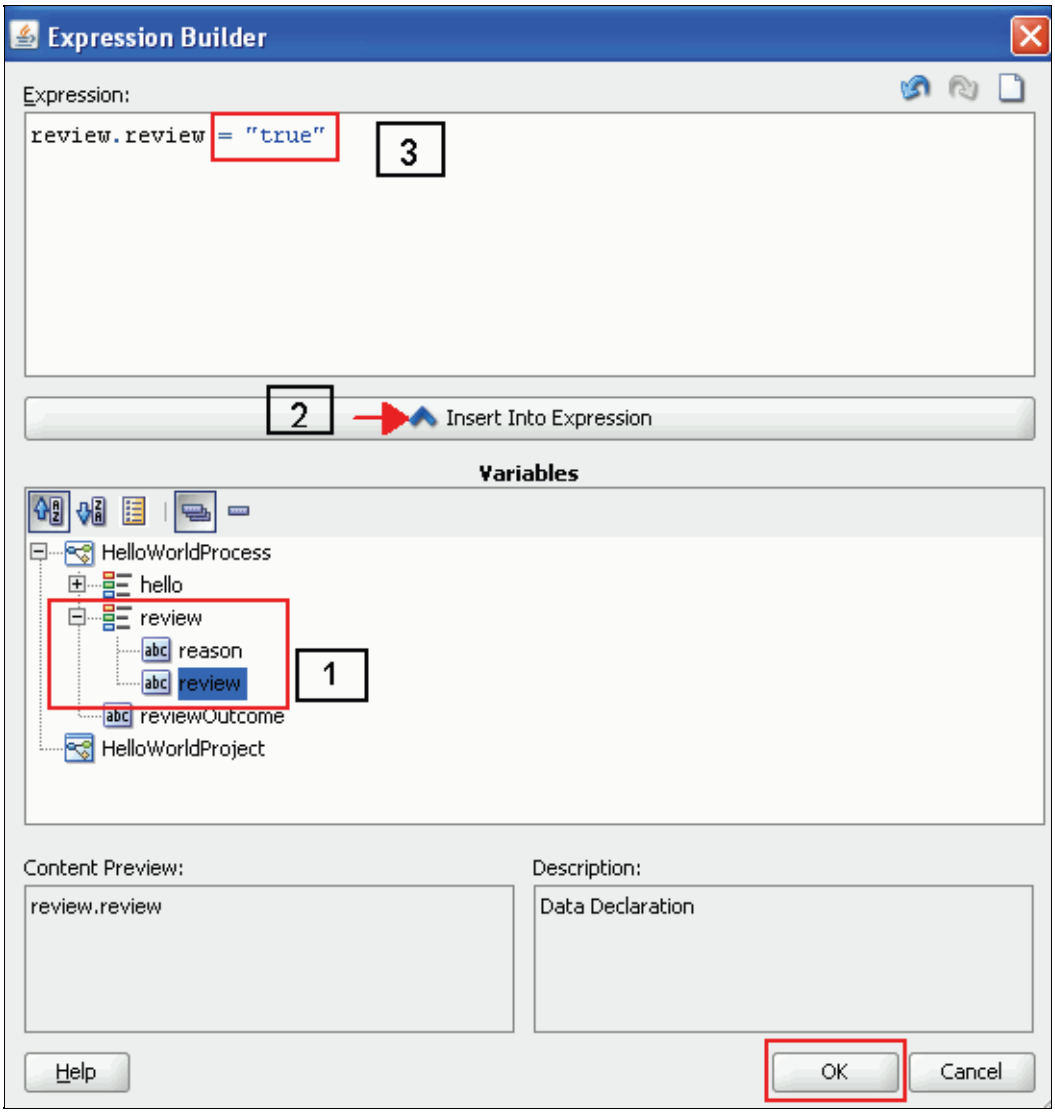
Click the Properties tab. On this tab, you define an expression whose outcome determines whether the process flows down this transition to the Review Message activity. In other words, you define the condition for the conditional sequence flow. Click the Expression Builder button on the right side of the window.



The Expression Builder opens. Build the expression in 3 steps:

1. In the Variables panel, expand **review** and select the **review** attribute.
2. Click the **Insert Into Expression** button. This puts `review.review` into the Expression panel at the top.
3. In the Expression panel, add:

   `= "true"` to the existing variable reference.

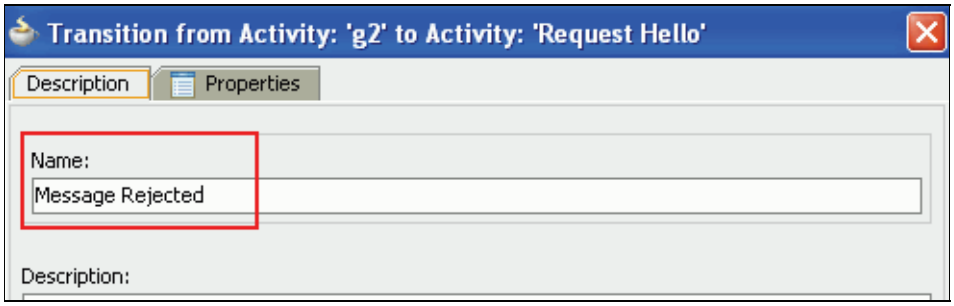


Click **OK** when finished to close the Expression Builder.

Click **OK** to close the Transition properties window.

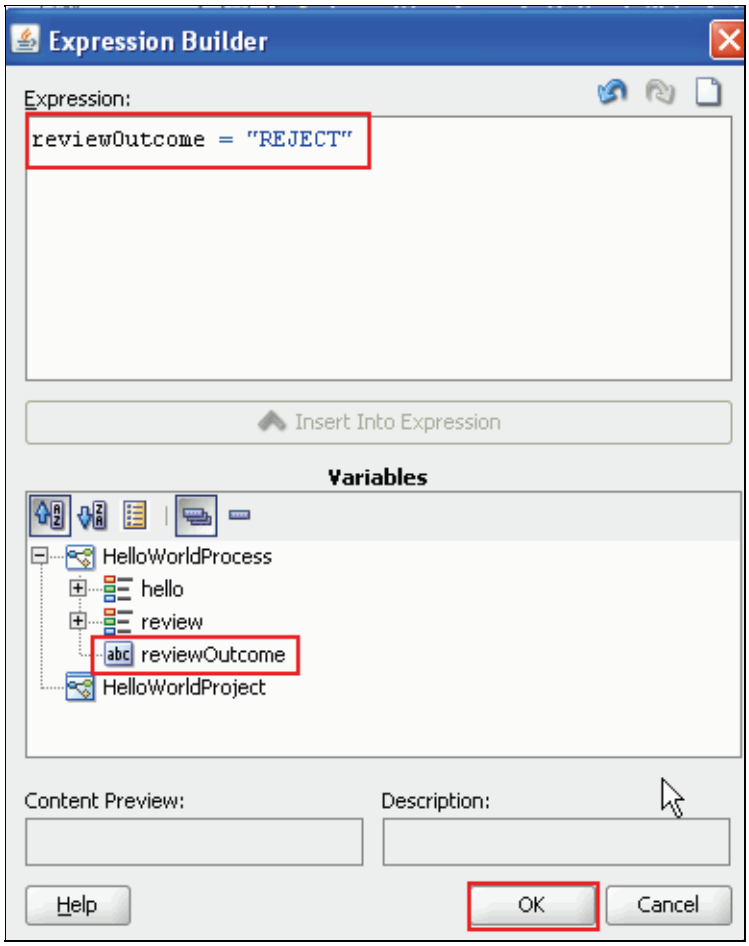
7. Define the condition for the sequence flow from **g2** to **Request Hello**. Double click on the transition line between the two objects. The Transition properties window opens.

   On the **Description** tab, name the transition `Message Rejected`.



Click the Properties tab. Click the **Expression Builder** button to open the Expression Builder and define the following expression, using the same procedure as you did in the last step:

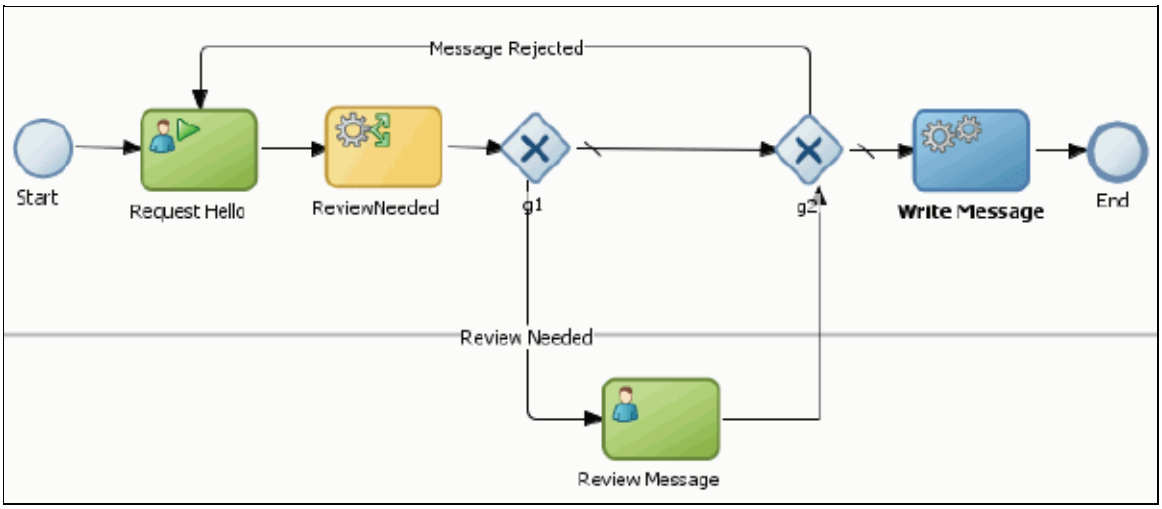
```
reviewOutcome = "REJECT"
```



Click **OK** to save the expression.

Click **OK** in the Properties window.

8. Your process should now look like this:



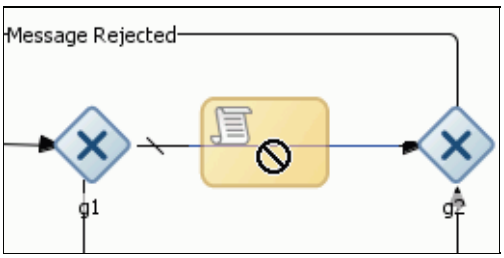
Adding a Script Task

1. When the process flow reaches the second gateway (g2), it checks the value of the `reviewOutcome` variable. There must be a value in the reviewOutcome variable in order for the process to move on to the Write Message activity. If the value is `REJECT`, the process returns to Request Hello. When a message is flagged for review, the

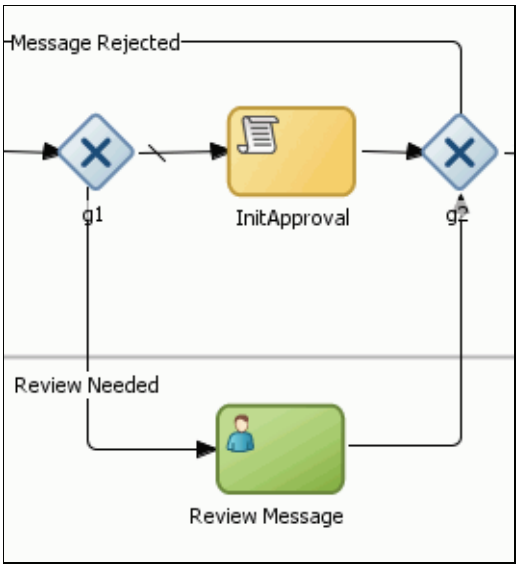
reviewer clicks either REJECT or ACCEPT, thereby populating the `reviewOutcome` variable.

If the process skips the review branch and moves from the g1 gateway directly to the g2 gateway, the `reviewOutcome` variable has no value.

Add a script task to initialize reviewOutcome between g1 and g2. From the Activities accordion panel of the Components Palette, click and drag a Script task, dropping it on the transition line between g1 and g2.

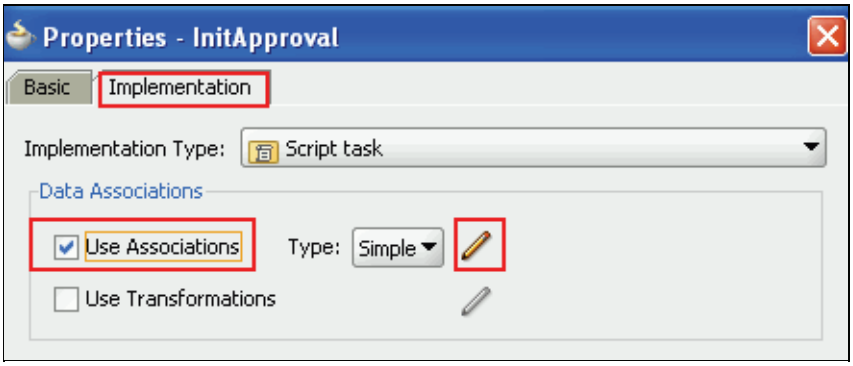


When the Properties window appears, name the activity **`InitApproval`** on the Basic tab and click **OK**.

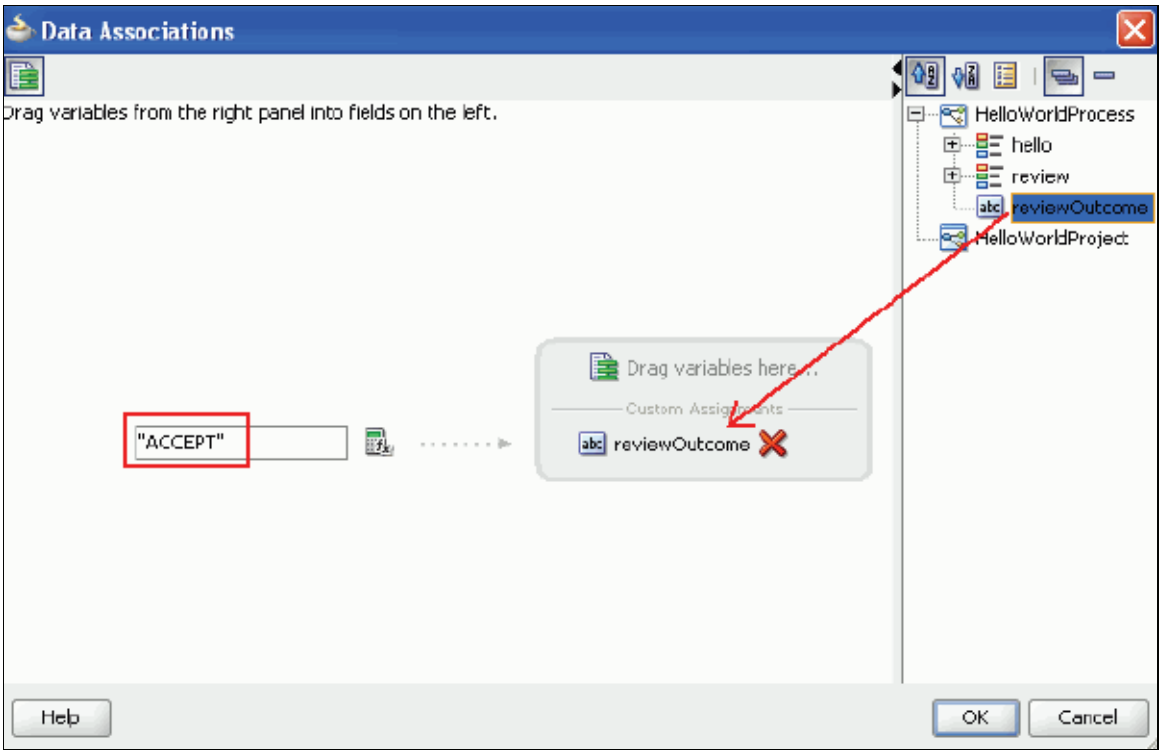


2. Define the implementation for the InitApproval task. Right click on **InitApproval** and select **Properties**.

   When the Properties window opens, click the **Implementation** tab. Then select the **Use Associations** checkbox. Click the pencil icon to open the Data Associations window.



In the Data Associations window, drag the **reviewOutcome** process data object from the right panel into the gray box in the left panel labeled "**Custom Assignments**". In the text field to the left of the gray box, provide an initial value for the `reviewOutcome` variable of **`"ACCEPT"`**.



Click **OK** in the Data Associations window.

Click **OK** in the Properties window.

Click **Save All**. The Hello World process is now complete!

Deploying and Testing the Application

In this section, you deploy the Hello World application to the BPM engine running in the WebLogic server that is part of your SOA installation. This tutorial assumes that your server is running on a remote Linux machine. You need to know:

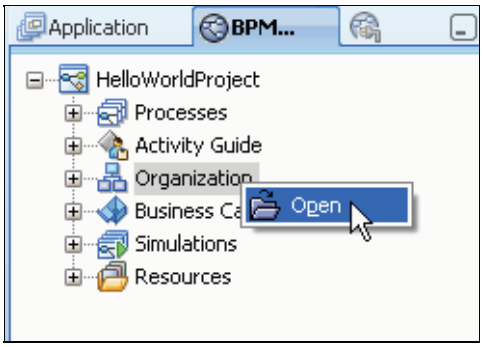
The server hostname and port
The WebLogic domain in which SOA is running
The username and password for the WebLogic administrative user
The name of the specific WebLogic server to which you will deploy. This tutorial assumes a single server (i.e. not managed instances) configuration and will deploy to the Admin Server.

Before deploying the application, you connect to the internal LDAP realm within the WebLogic server and map the Reviewer and Requester roles to a user in the LDAP.

After deployment, you run the **Oracle BPM Workspace** web application to test the Hello World application.

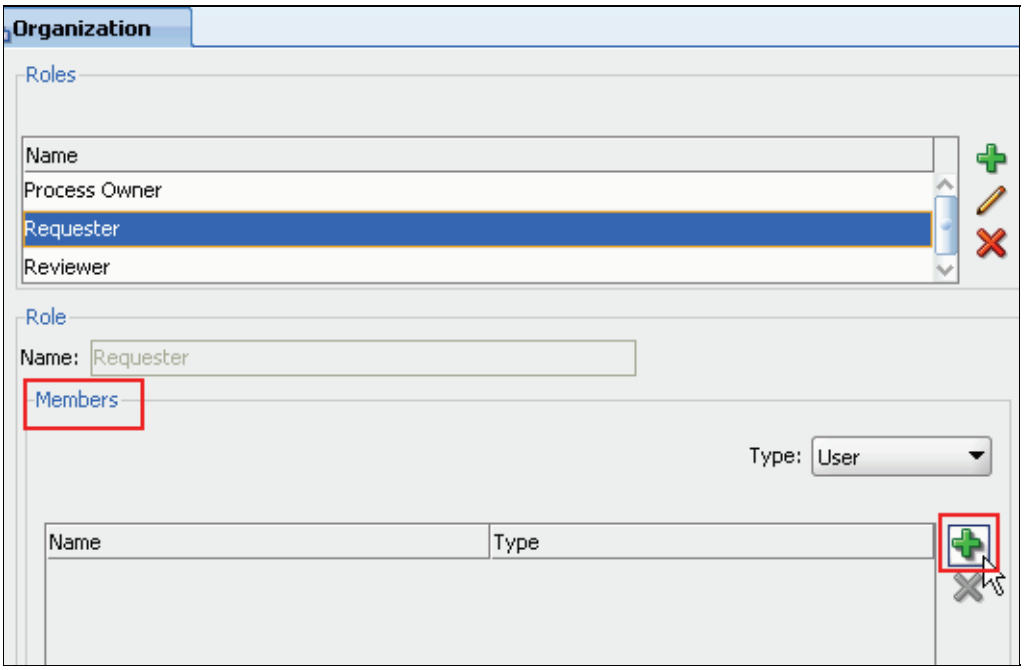
Mapping the Studio Role to an LDAP Role

1. In the **BPM Project Navigator**, expand the **HelloWorldProject** and right click on the **Organization** node, selecting **Open**.



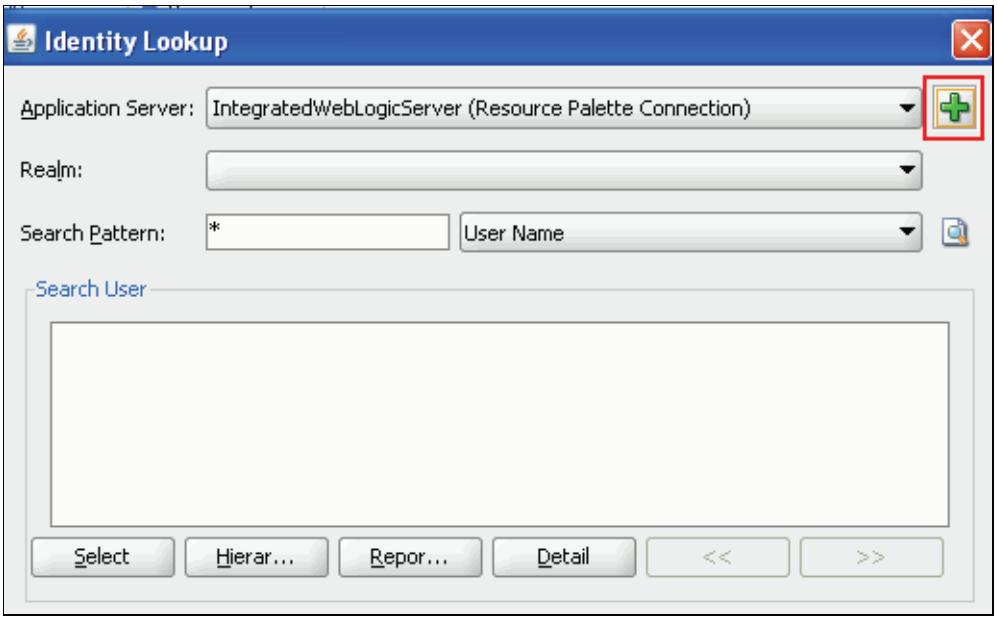
It may take several seconds to open the Organization editor.

2. Add user members to the Requester role. In the **Organization** editor, select the **Requester** role. Click the green plus sign button to the right of the **Members** panel.

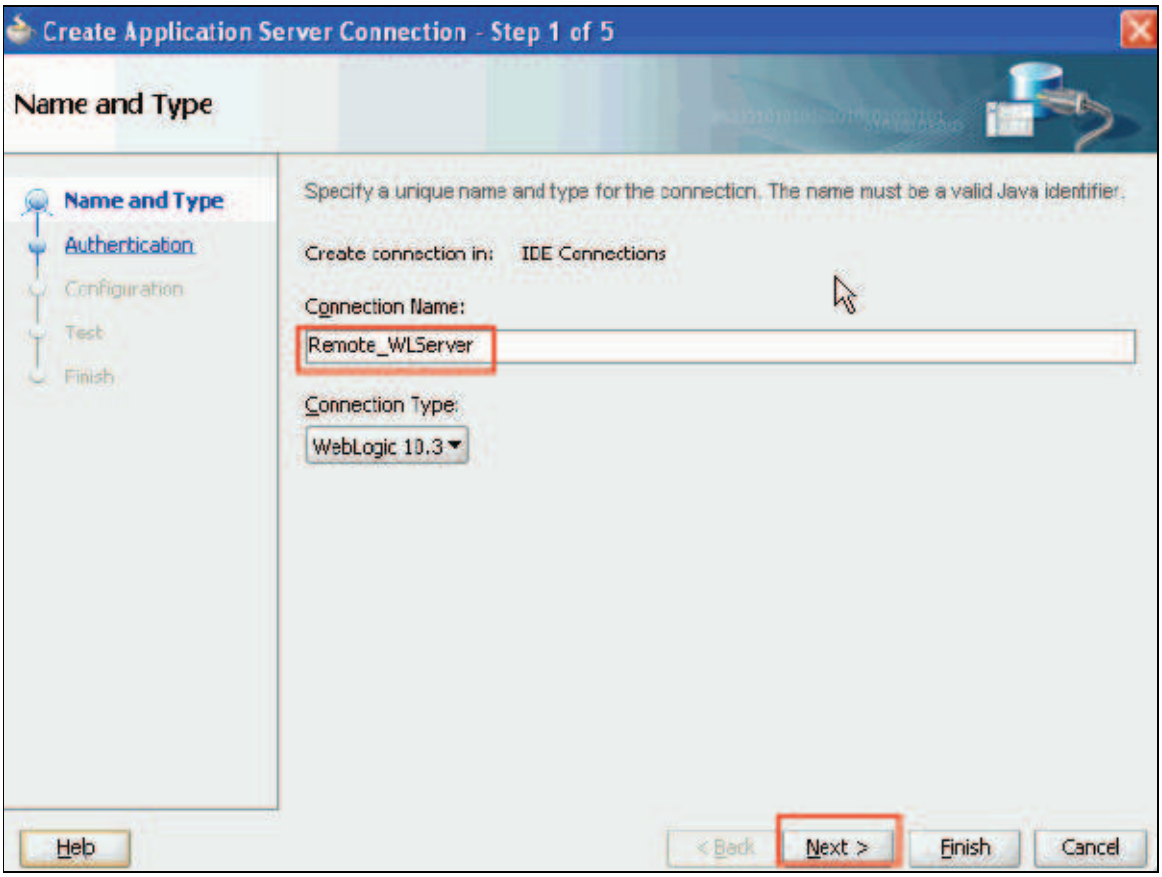


The **Identity Lookup** window opens. Studio is not yet aware of your remote server, so the only application server that appears is the integrated weblogic server that was installed with JDeveloper.

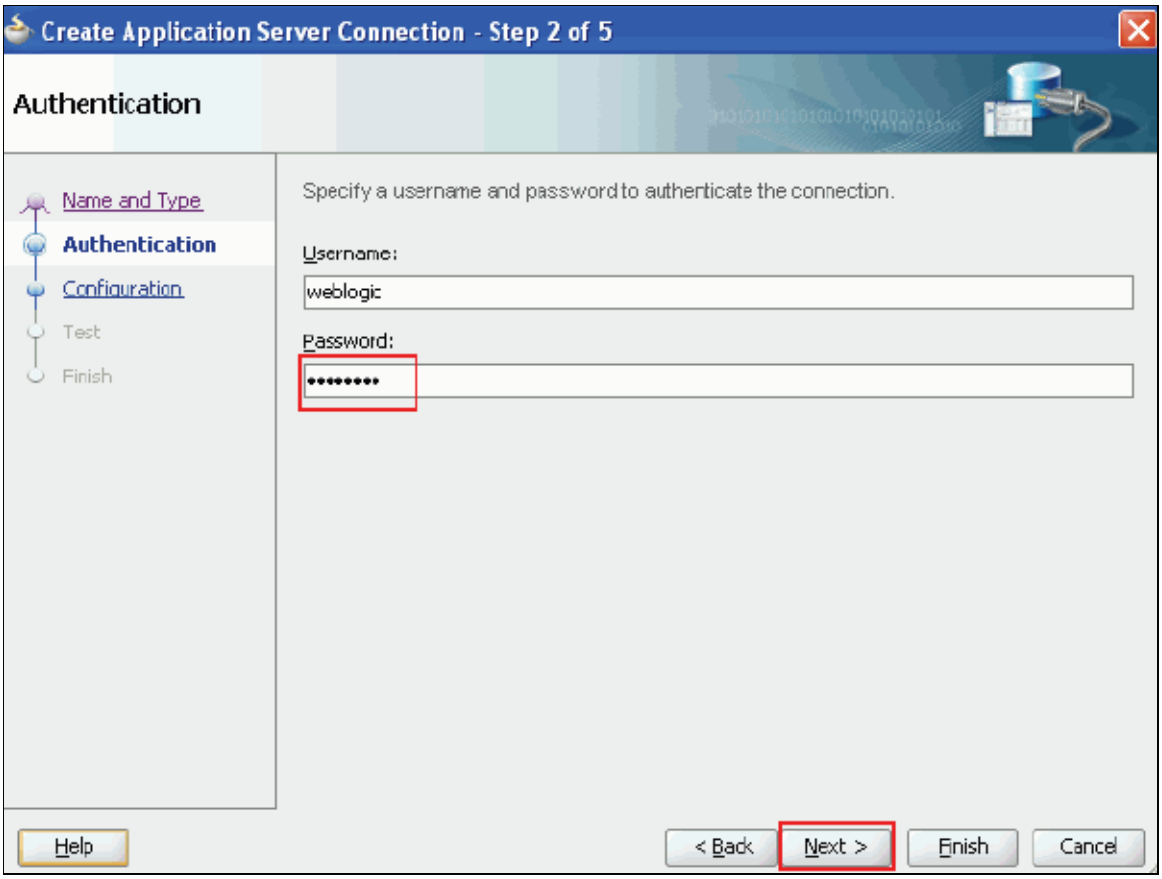
Create a new profile for your remote server. Click the green plus sign icon next to the **Application Server** field to launch the **Create Application Server Connection** wizard.



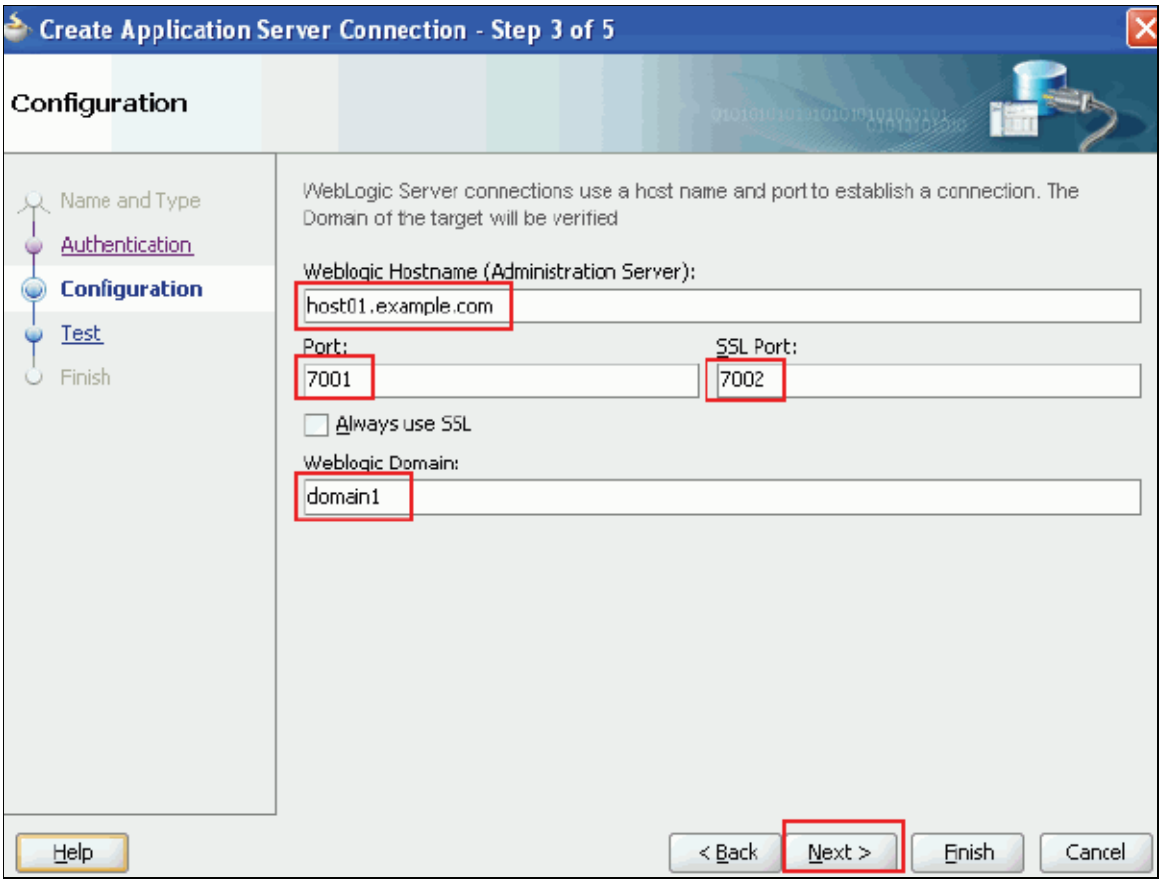
3. In the Name and Type page of the wizard, enter `Remote_WLServer` as the **Connection Name**. Make sure that the **Connection Type** is `WebLogic 10.3`, and click **Next**.



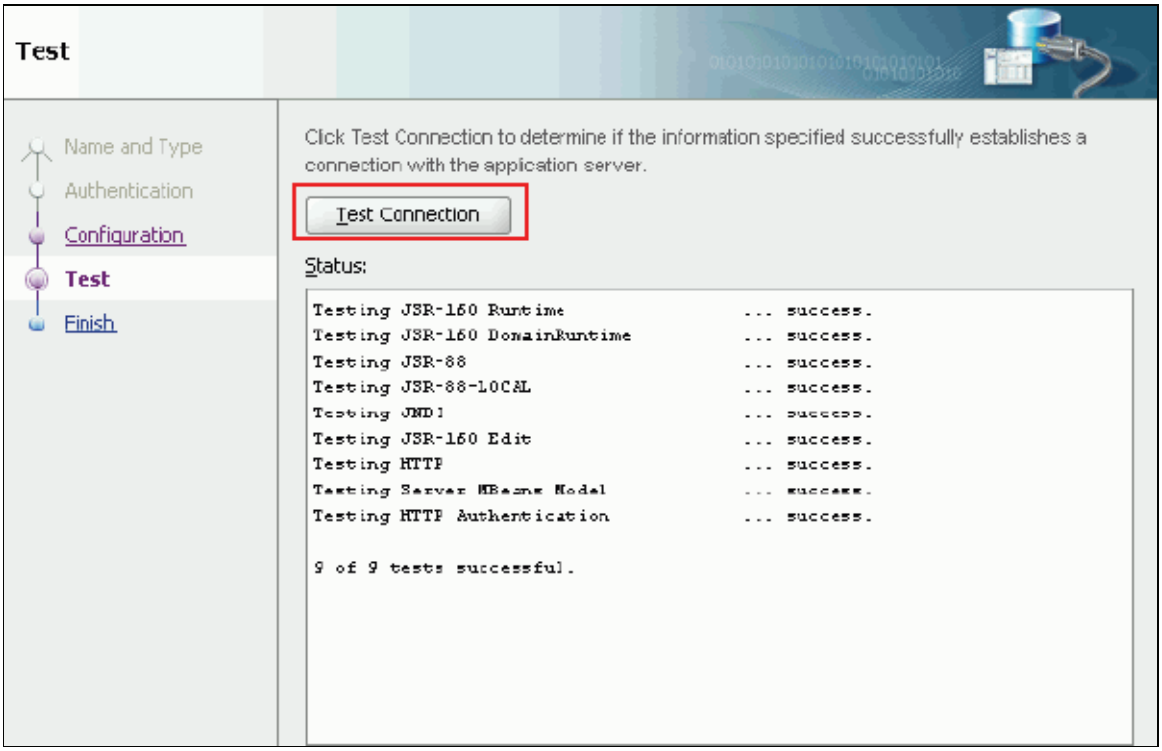
In the Authentication page of the wizard, enter `weblogic` as the **Username** and `welcome1` as the **Password** (or substitute your weblogic password). Click **Next**.



In the Configuration page of the wizard, enter your **Weblogic Hostname**, accept the default values for Port and SSL Port, and enter `domain1` in the **Weblogic Domain** field. Click **Next**.



In the Test page of the wizard, click the **Test Connection** button. You should see results similar to those shown below. If not, click the **Back** button, correct any errors and test again.

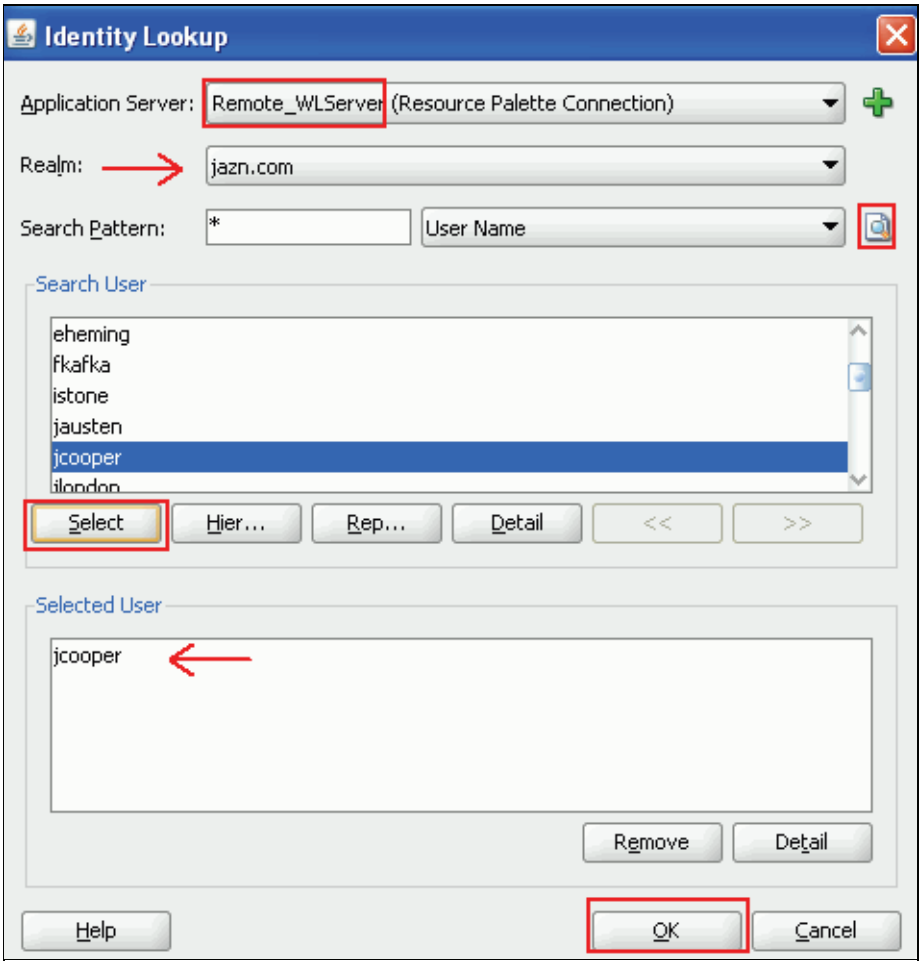


Click **Finish**.

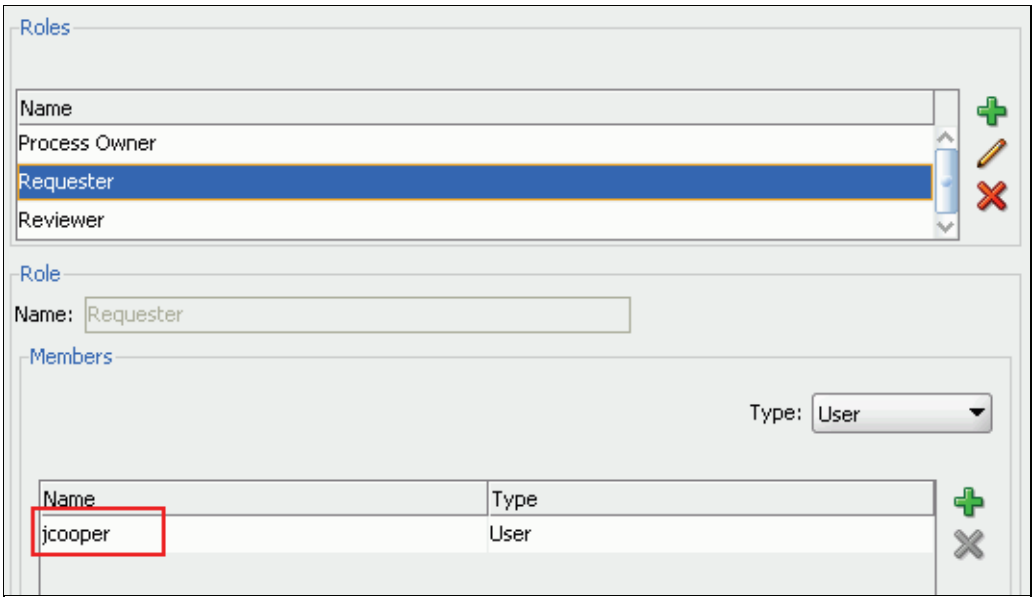
You are now returned to the Identity Lookup window.

4. Select the **Remote_WLServer** profile from the **Application Server** list. After a few seconds, Studio will connect to the LDAP server in the remote server and you see **jazn.*** appearing in the **Realm** field.

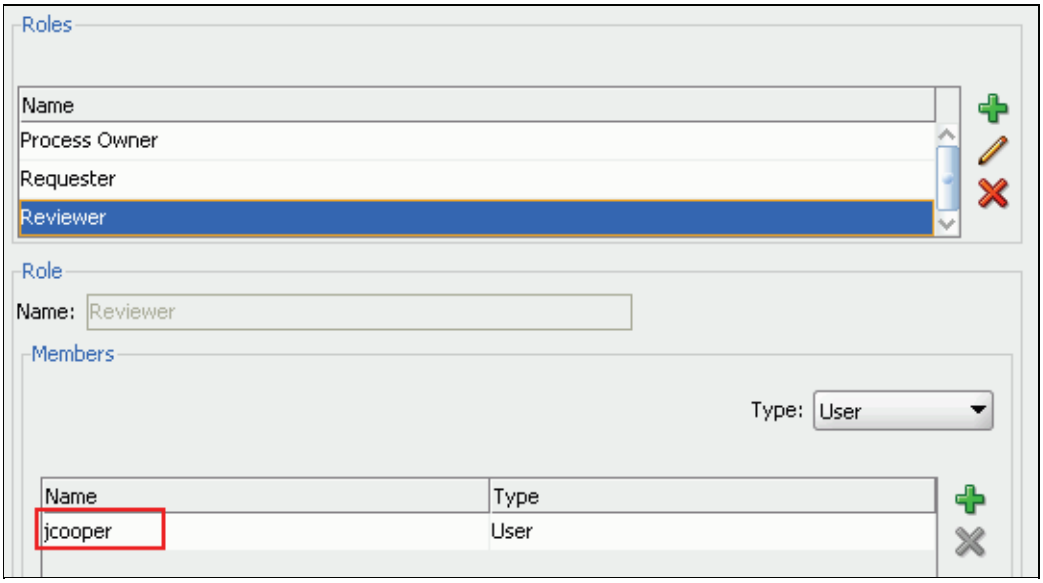
   Click the browse icon next to the **User Name** field to bring up a list of all users in the Demo Community from the LDAP server. Select **jcooper** from the list and click the **Select** button.



   `jcooper` will appear in the Selected Users panel of this window. Click **OK**. You are returned to the Organization editor.



5. Follow the same procedure as in the previous step to add `jcooper` to the **Reviewer** role as well.

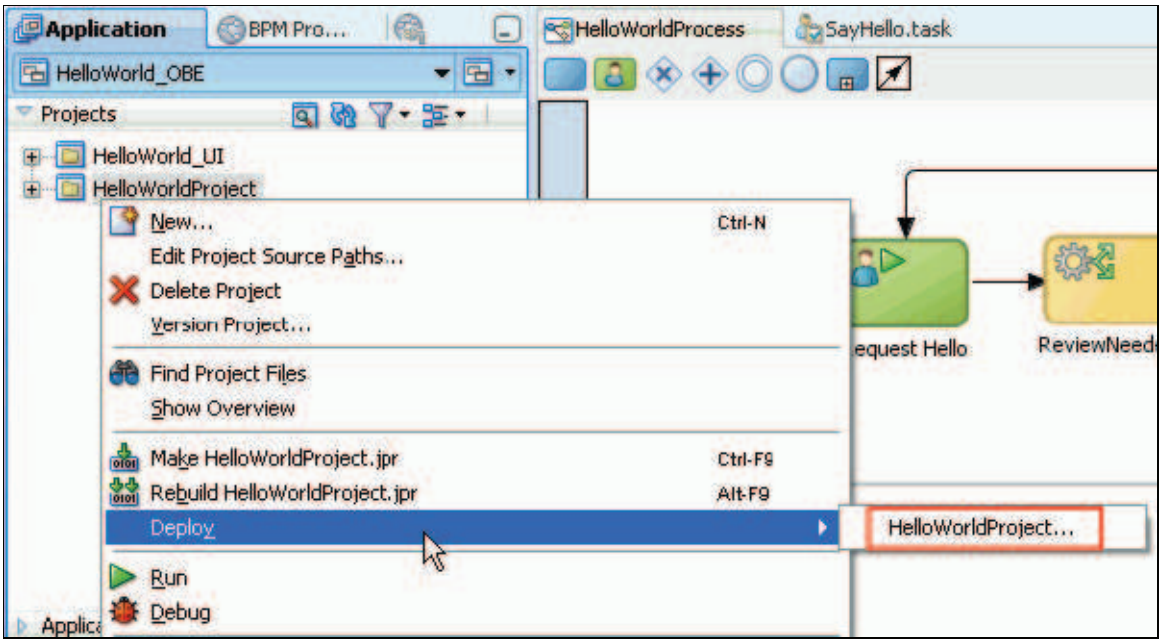


6. Click **Save All** and close the Organization editor.

Deploying the Process

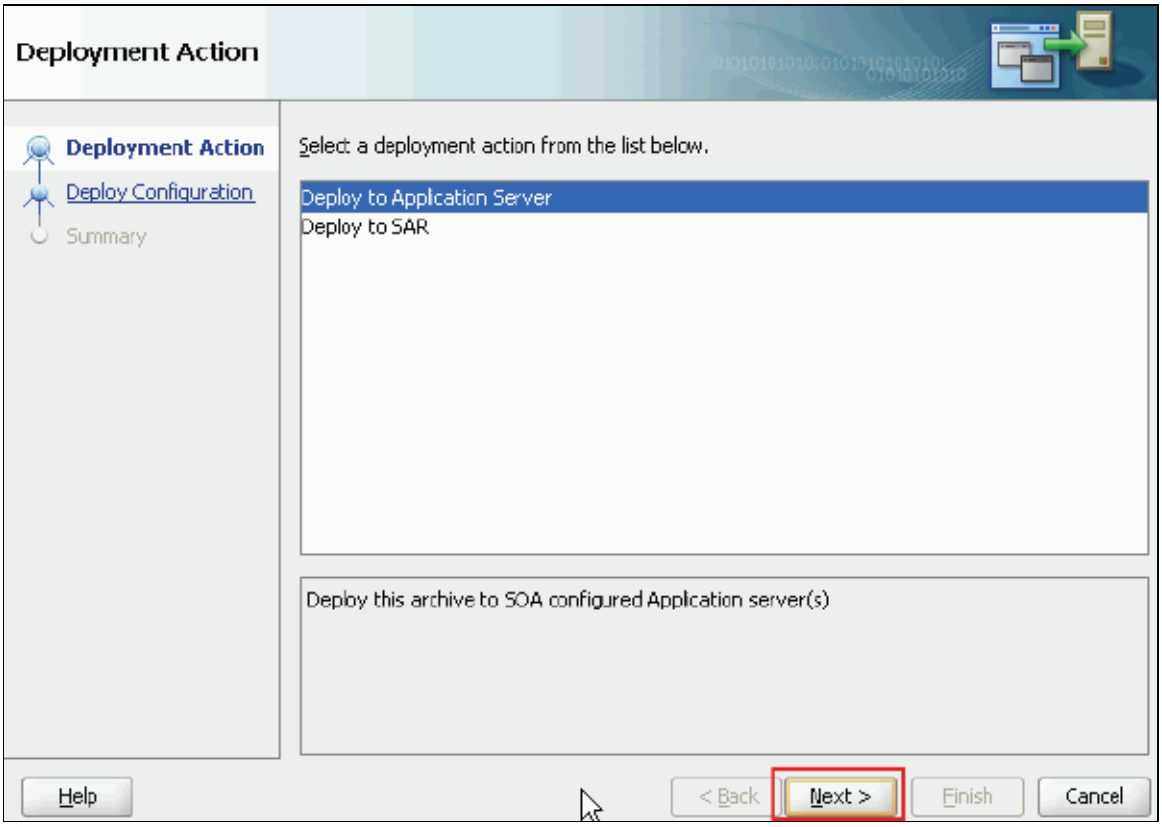
1. Deploy the HelloWorldProject.

   In the Application Navigator, right click **HelloWorldProject** and select **Deploy > HelloWorldProject...**

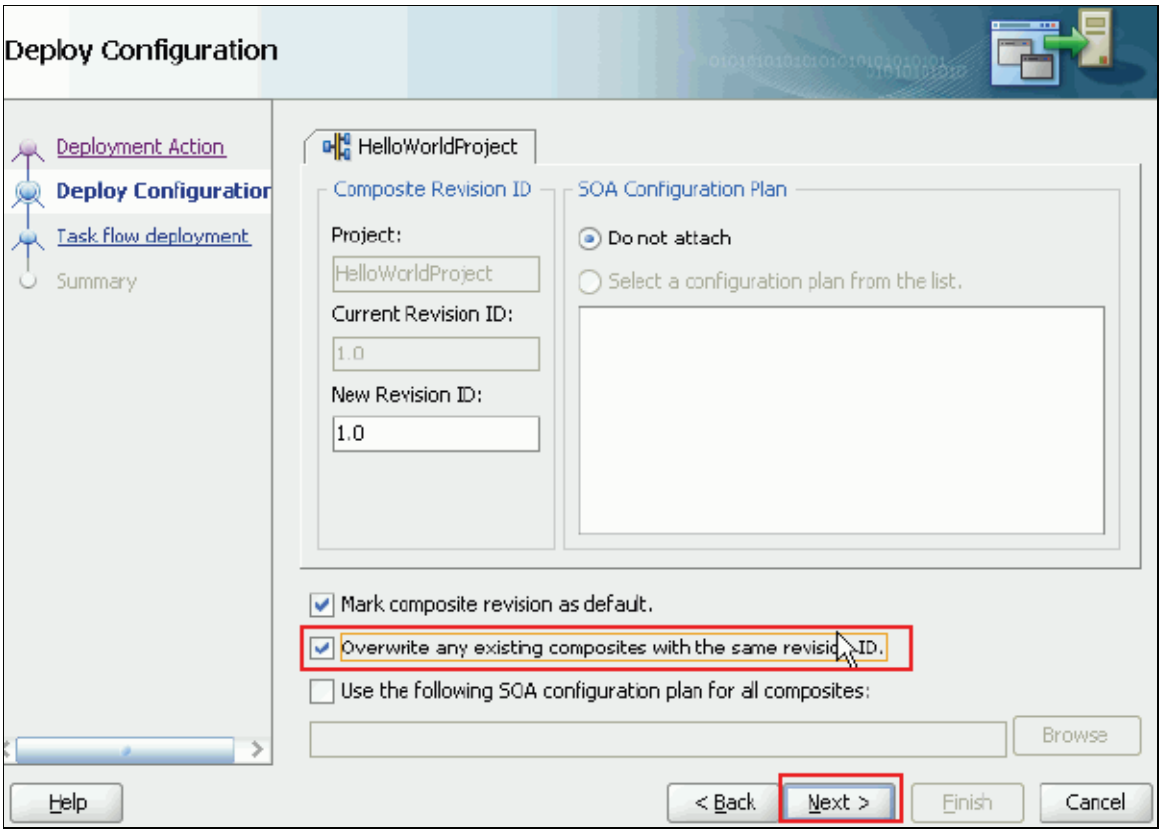


The **Deploy HelloWorldProject** wizard opens.

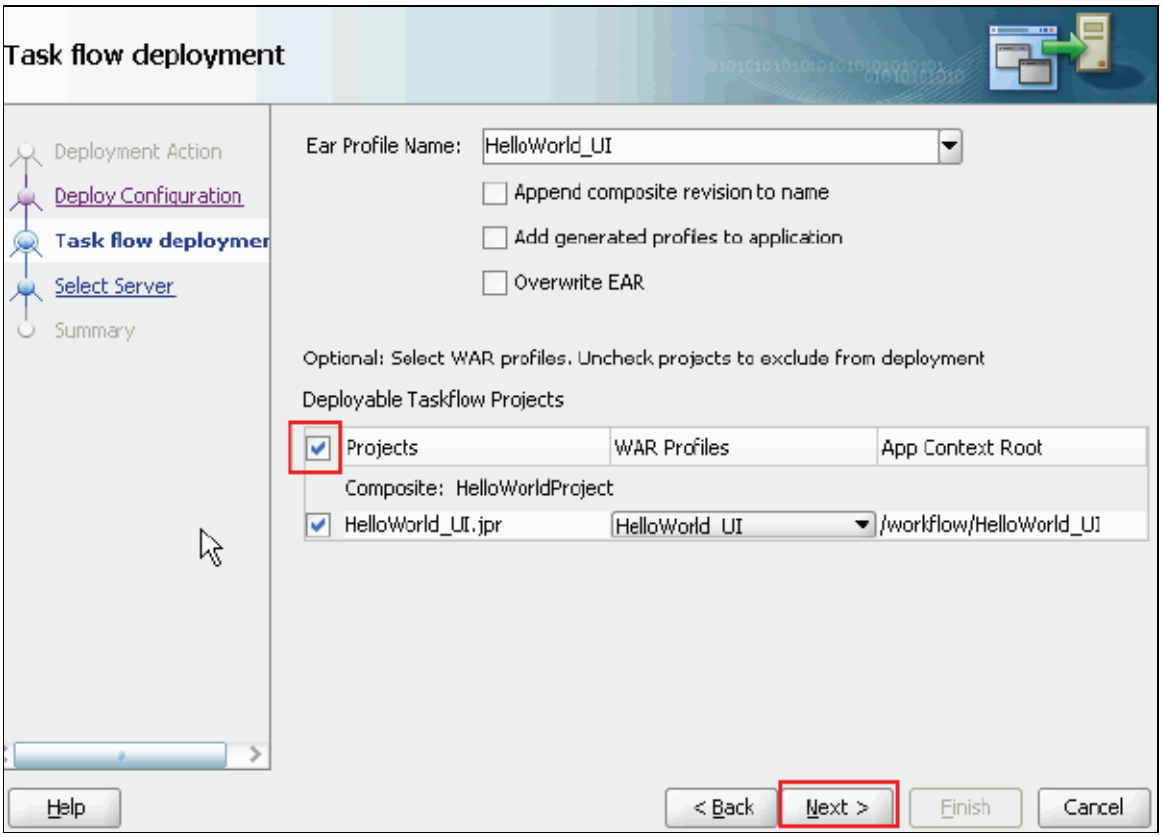
2. In the Deployment Action page of the wizard, select **Deploy to Application Server** and click **Next**.



In the Deploy Configuration page, click the **Overwrite any existing composites with the same revision ID** checkbox and click **Next**.

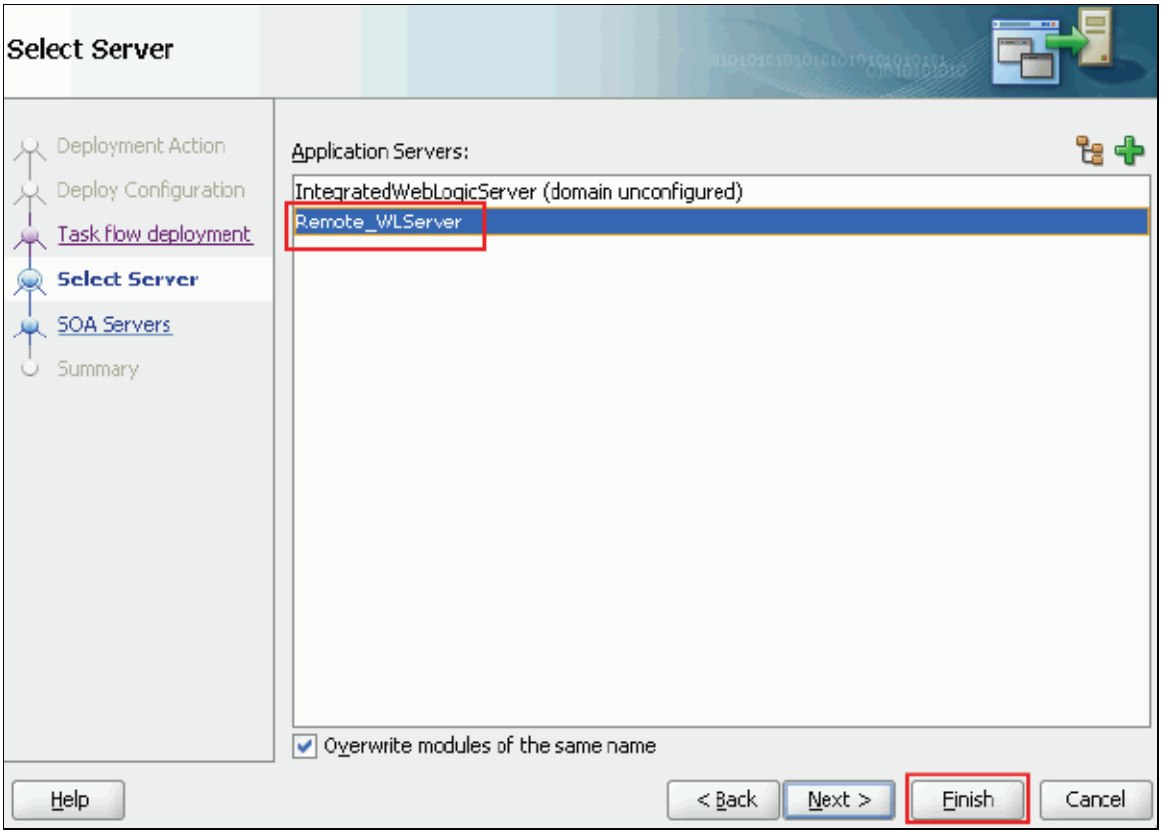


In the Task flow deployment page, select the checkbox next to **Projects**. This selects all taskflow projects in the HelloWorldProject, so you will see the HelloWorld_UI project selected now also.

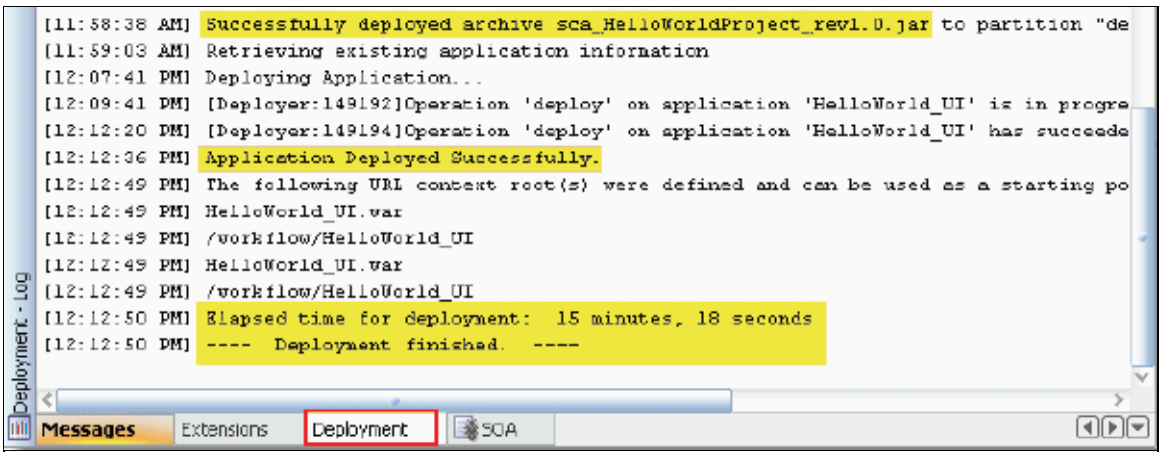


Click **Next**.

In the Select Server page, select **Remote_WLServer** and click **Finish**. Deployment will begin.



Check the **Deployment** tab in the **Log** panel in the lower central portion of the Studio window to watch the progress and determine when deployment has finished. This first deployment of the SOA Composite (shown as sca_HelloWorldProject_rev1.0.jar in the log viewer screenshot below) will be very quick. The deployment of the WAR file containing the taskflows (HelloWorld_UI.war) will take much longer. Total deployment time will be around 15 minutes, depending upon your environment.



Testing the Process in Workspace

1. Open a browser (either on the server or from your windows machine) and enter the following URL:

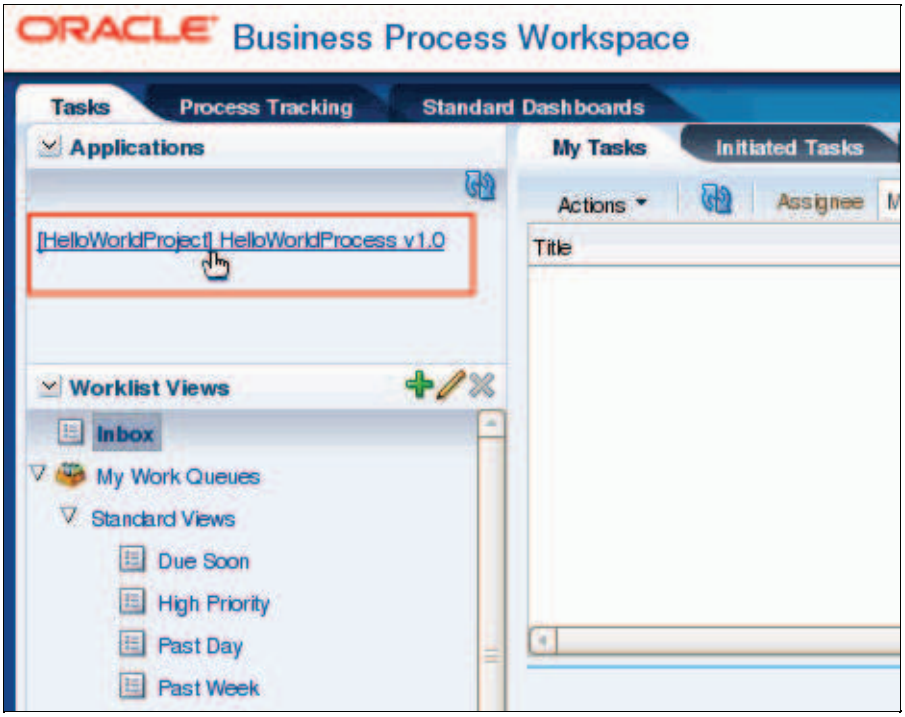
   `http://`*<your server hostname>*`:7001/bpm/workspace`

   When the Welcome page of the Workspace appears, enter `jcooper` in the **Username** field and `welcome1` in the **Password** field. Click **Log In**.

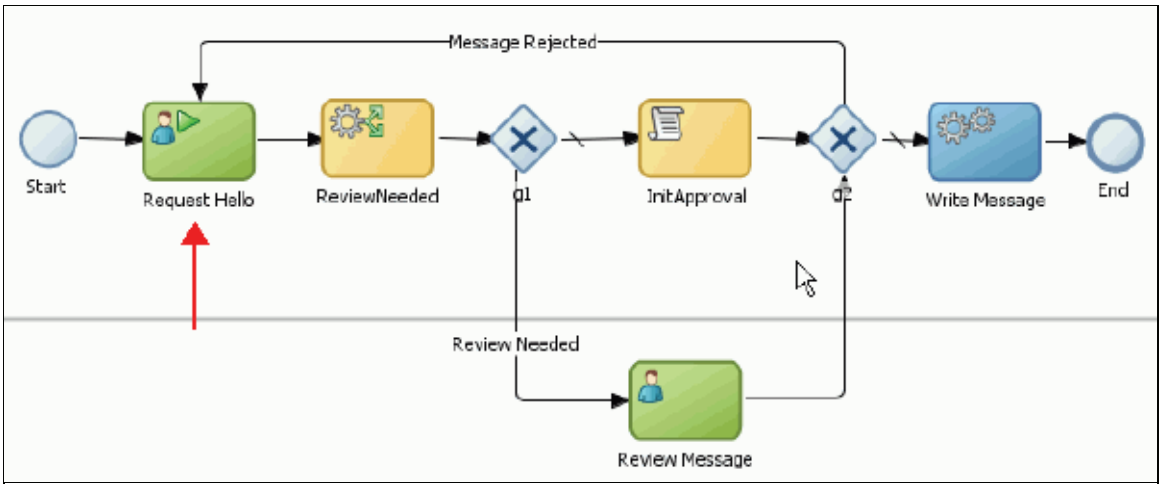
The main Workspace window opens.

2. Instantiate the process by clicking the **HelloWorldProcess v1.0** link under **Applications** in the **Tasks** tab on the left side of the window.

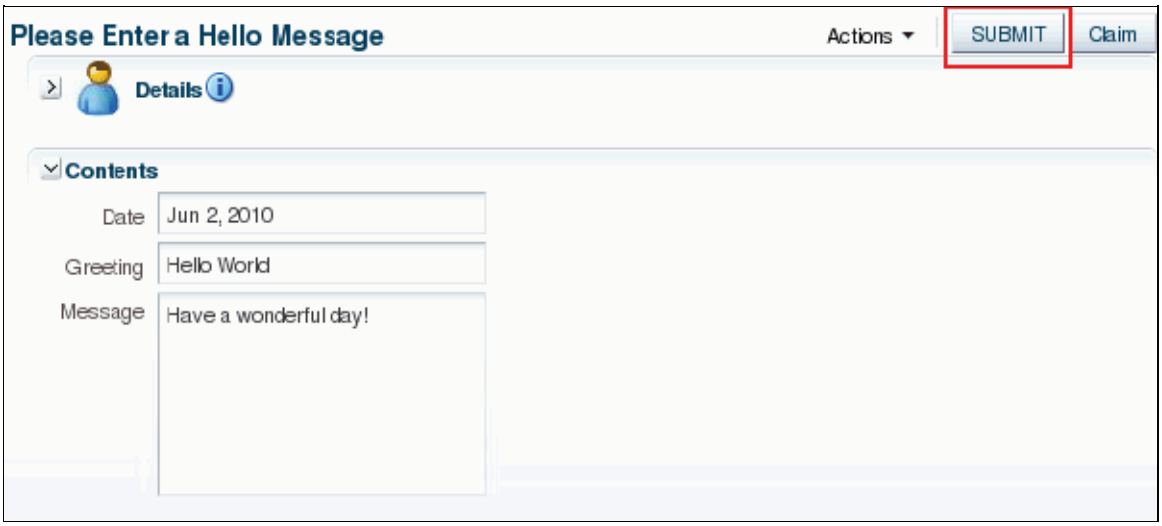
   Note: If the HelloWorldProcess v1.0 link does not appear in the Applications panel on the Tasks tab, click the **Process Tracking** tab. It should appear there. You can instantiate the process from that tab.



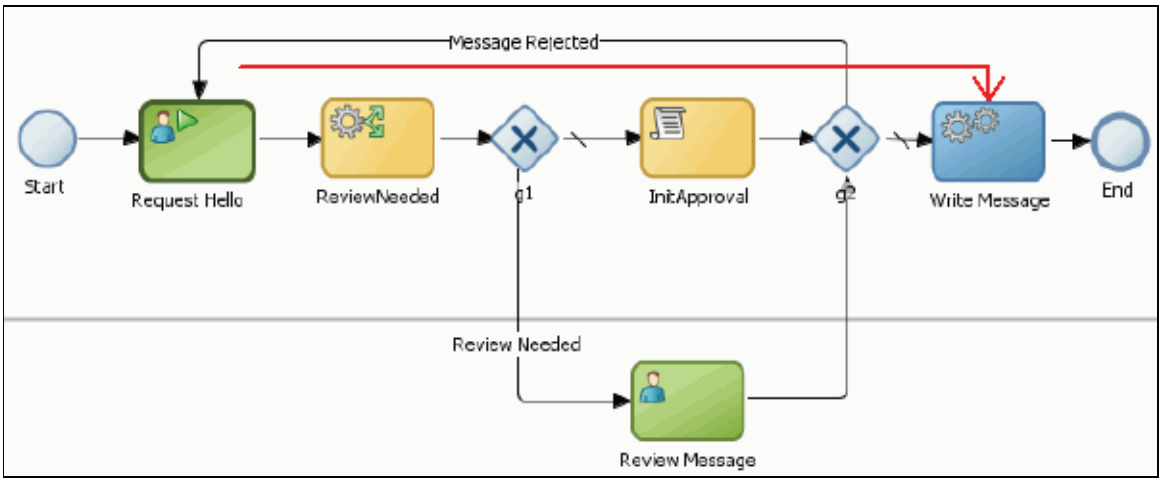
This action executes the first task of the process, **Request Hello**.



3. First enter a message that will *not* require review. In the **Please Enter a Hello Message** popup window that appears, enter a value in the **Date** field using the format `MMM d, yyyy` (example: Jun 2, 2010) Also enter a value in both the **Greeting** and **Message** fields (Greeting field length > 5, Message field length > 5). Click **Submit**.



After submitting the message, the process goes to the ReviewNeeded business rule. Since both the Greeting and the Message are considered "Medium" in length, the message does not require a review. Hence the process goes through the `g1` gateway, the `reviewOutcome` variable is initialized in the script task, and the process flows on through the `g2` gateway to the **Write Message** activity.



The Write Message activity is a service type activity, not an interactive activity., therefore there will be no indication

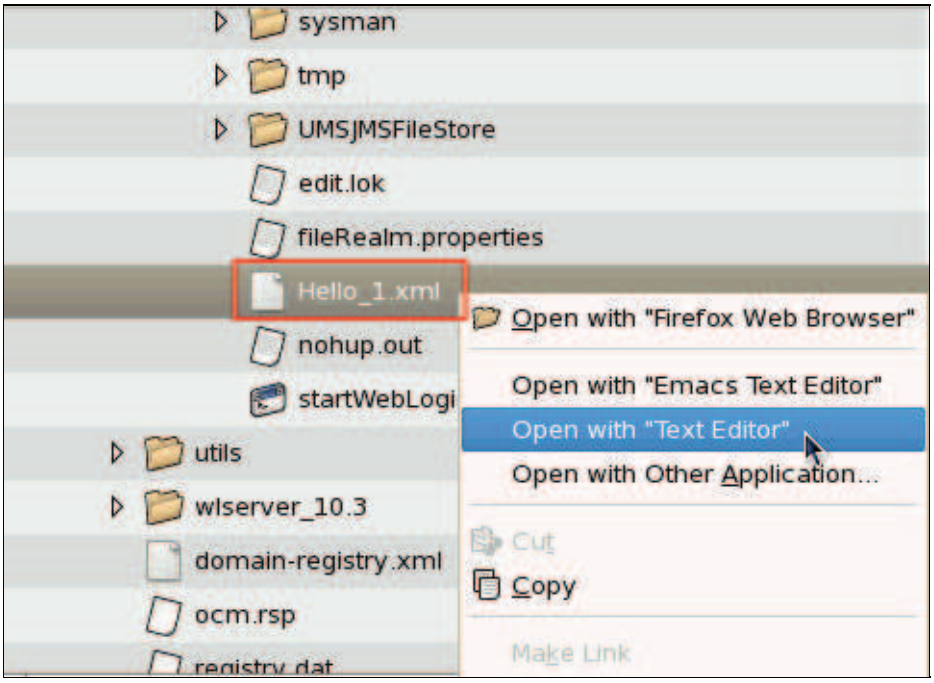
of its action within the Tasks panel of the Workspace.

4. Open the file that was created by the Write Message implementation. Recall that you configured the file adaptor to write the file to the file path "**.**" (dot). This is relative to the `domain1` directory within the WebLogic file structure.

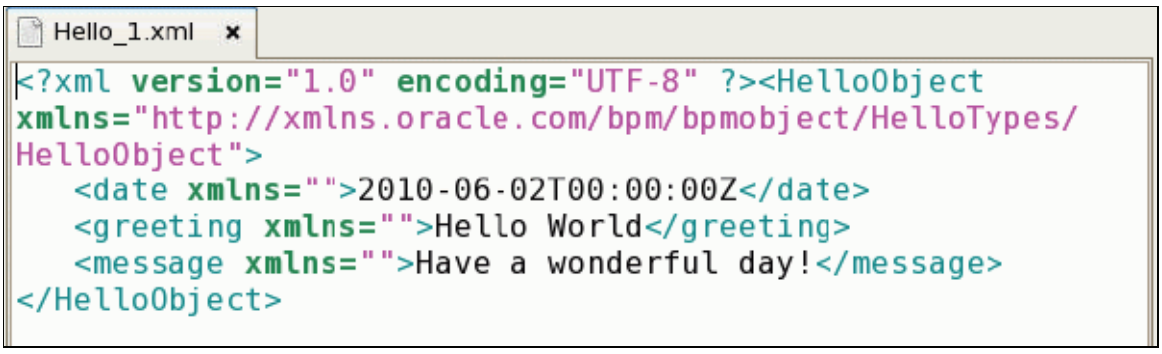
   On the server, using either a terminal window or an Explorer style window as shown here, navigate to:

   <path to your mwhome>/**mwhome/user_projects/domains/domain1**

   Find the file `Hello_1.xml`. Open it in a text editor to view the outcome from your process.

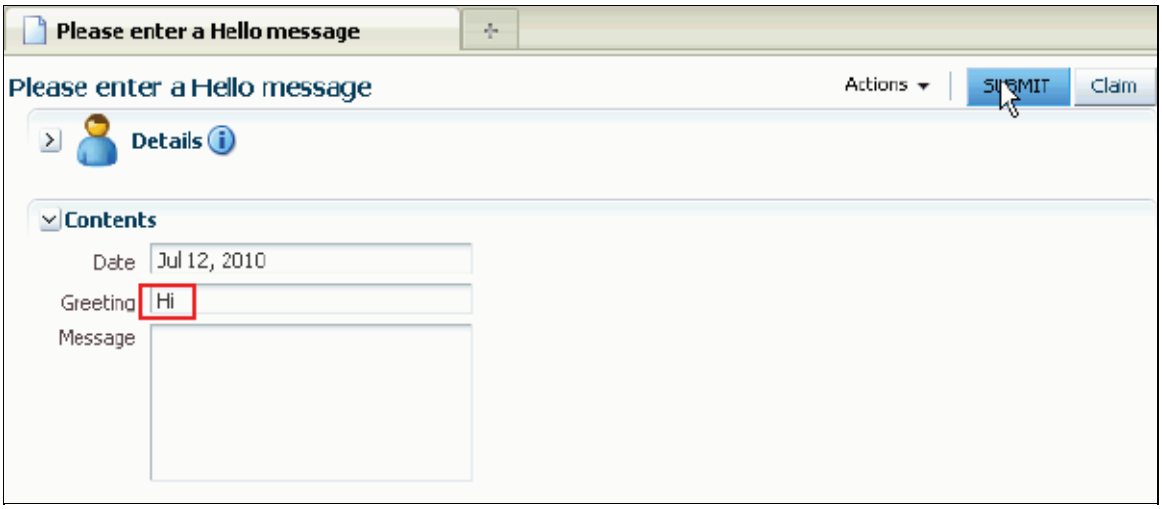


   It should look something like this:

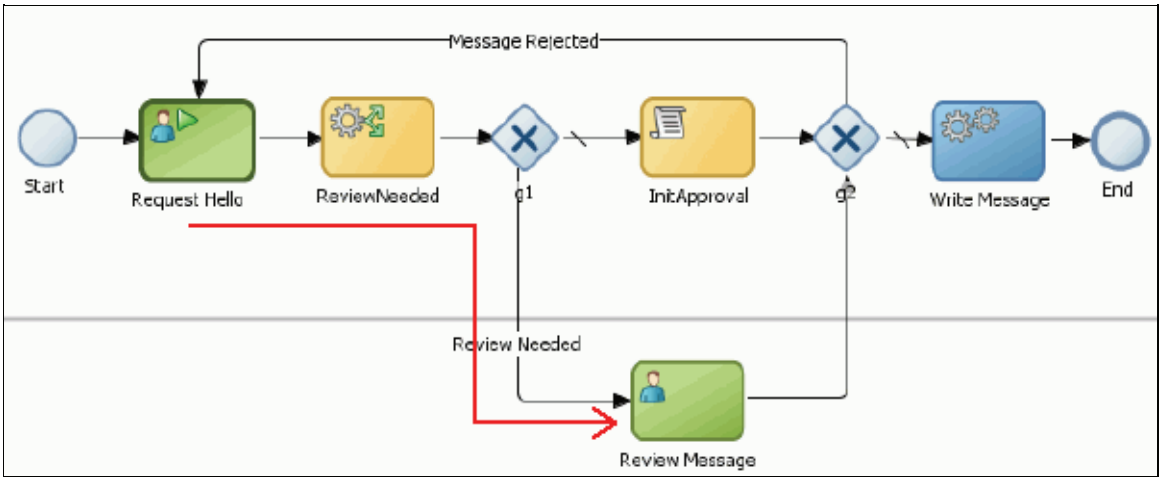


5. Create another instance of the HelloWorld process by once more clicking the HelloWorldProcess v1.0 link. This time, enter a message that will be routed to the Review Message activity.

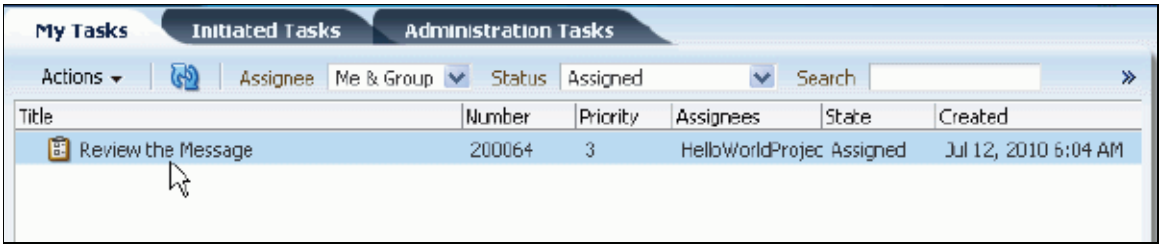
   When the **Please enter a hello message** window appears, enter a date and some text that has a *length* < *5* in the **Greeting** field. Leave the **Message** field blank. Click **Submit**.



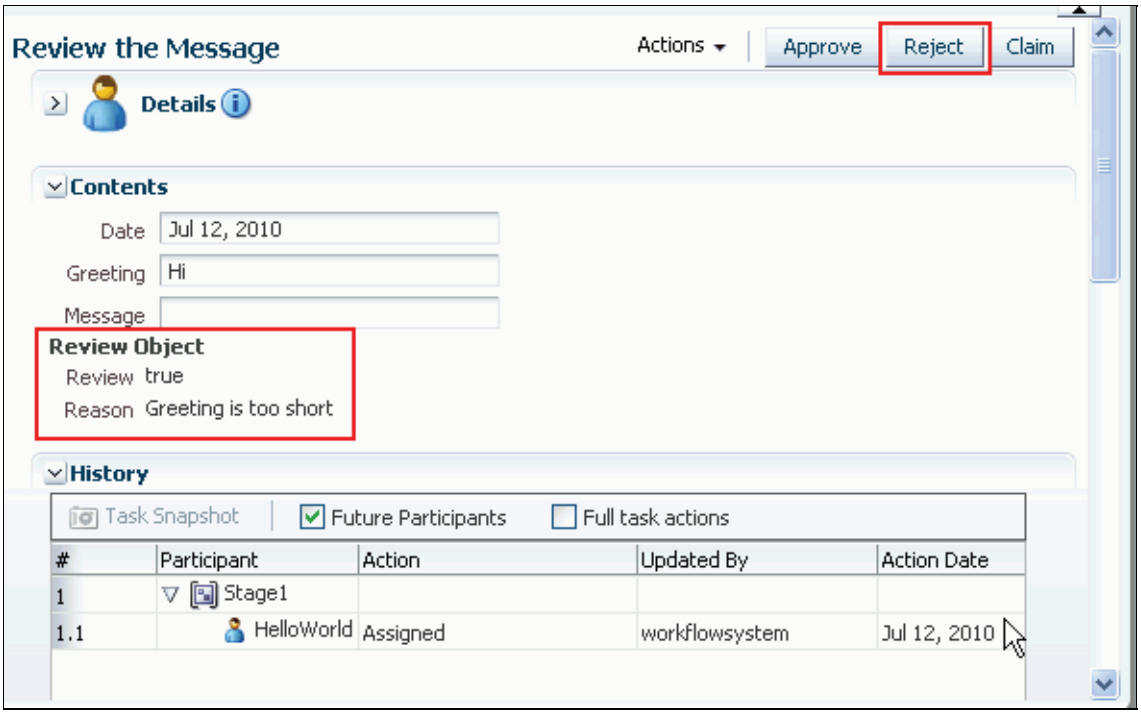
6. The process now flows, as before, to the business rule where the greeting is found to be "Short". As you'll recall, when the greeting is "short", the message requires review, regardless of what is in the Message field. Consequently, the process now flows to the Review the Message activity, which is assigned to the Reviewer role.



The **Review the Message** activity now appears in the **My Tasks** tab (remember that jcooper also has the **Reviewer** role). Click it to see the details in the lower panel of this tab.

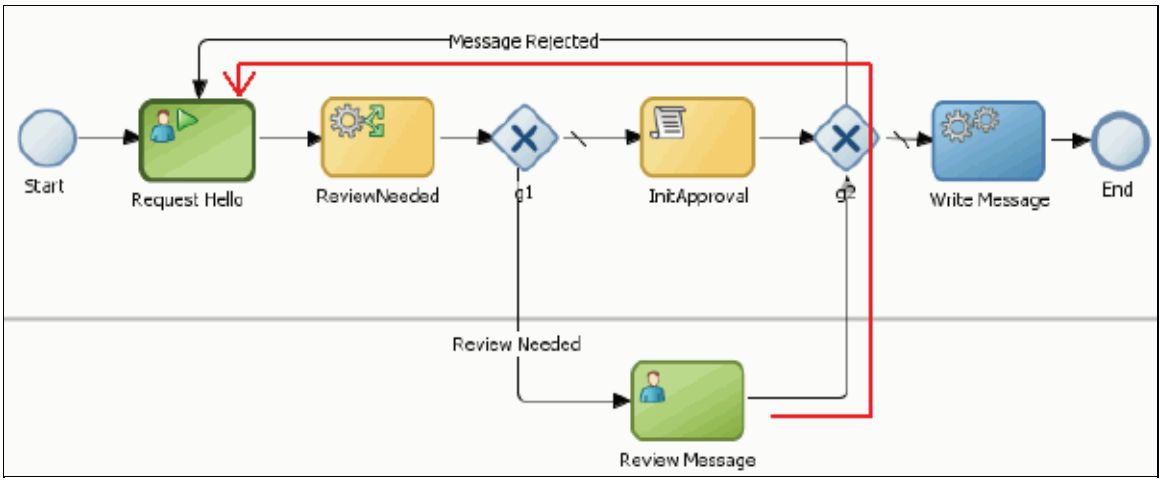


In the details panel, you can see the taskflow form that you created for this activity, containing form elements for both the HelloObject and the ReviewObject. Notice that the ReviewObject attributes have been assigned by the Business Rules engine after executing the business rule for this message.

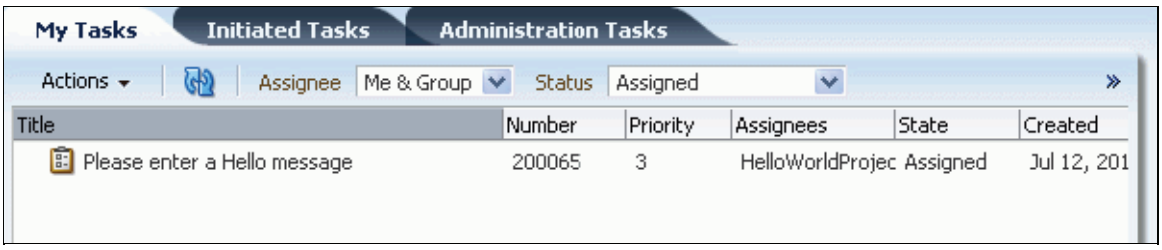


Click **Reject** when you've finished examining the details.

7. The process now flows to the `g2` gateway where the value of the `reviewOutcome` variable is examined. It has been set to `REJECT` by the reviewer, so the process now flows back to Request Hello.



The **Please enter a hello message** link appears in the My Tasks panel. This is the Title string for the human taskflow associated with the **Request Hello** task. Previously, this window was launched immediately upon instantiation of the process. This time it appears within the My Tasks panel.



Click the link to view the input form and enter a more appropriate message that will *not* require review as you did in Step 3. Click **Submit** when finished.

The process will now flow through the two gateways to the Write Message activity and to the End activity. The My