ORACLE

BPEL Tutorial

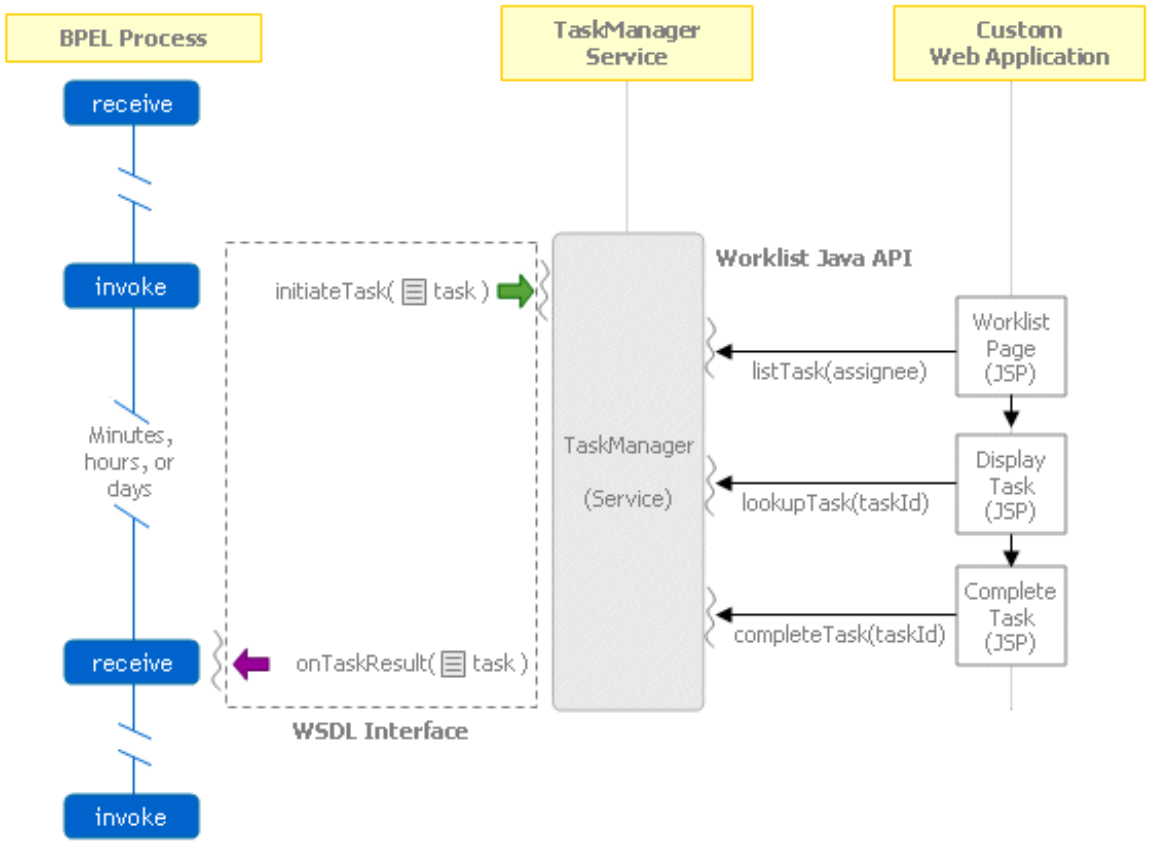
Tutorial 6: Working with the TaskManager Service

BPEL is a language for composing multiple services into an end-to-end business process. People and manual tasks are often an integral part of such business processes (particularly for exception handling or workflow/approval-related tasks). In this document, you will learn how to use the Oracle TaskManager service to model user interactions within a BPEL business process.

Contents

Introduction

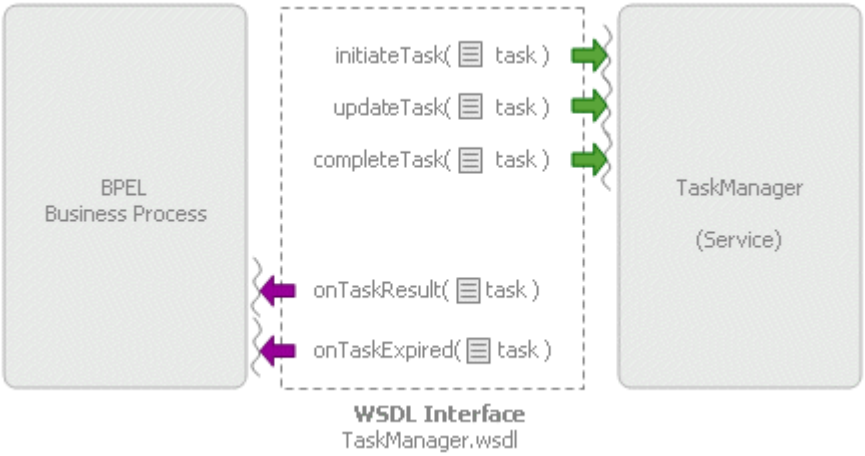
The TaskManager is a service provided by the Oracle BPEL Process Manager to help model user interactions (tasks, exception management, and so on) with a BPEL business process. The TaskManager offers a WSDL interface to enable a BPEL process to initiate a task and receive callback notifications when the task is completed. It also offers a Java Worklist API to enable developers to build GUI applications that list, display, and complete the tasks assigned to a specific user or group.



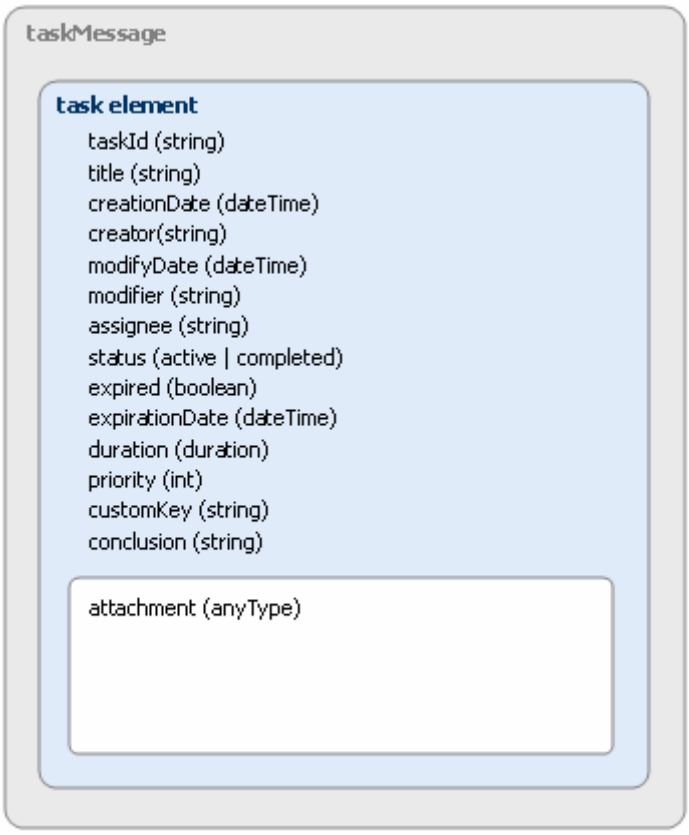
Following an overview of the TaskManager service in the next section, you will learn how to use the WSDL interface to integrate the TaskManager service into a BPEL process, looking first at the source code and then at how to use the BPEL Designer to create such code for you. You will then learn more about using the Java Worklist API to build a custom user interface for listing, viewing, and completing tasks.

Overview of the TaskManager Service

The TaskManager service is an asynchronous service that manages the lifecycle of a task. It offers a WSDL interface that enables a BPEL process to initiate tasks, and a callback interface for notifying a process of the completion or expiration of a given task. The WSDL interface also enables a BPEL process, when necessary, to update a task (for example, to reassign it to another user) or to complete or cancel a task.



As illustrated below, a task document captures all the information about a task, and a taskMessage is a simple wrapper around a task document. All inbound and outbound operations use the same taskMessage to exchange information about a task.



From `Task.xsd` (at http://localhost:9700/orabpel/default/TaskManager/Task.xsd), you can see that the task document type is as follows:

```
<xs:element name="task">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="taskId" type="xs:string" minOccurs="0"/>
      <xs:element name="title" type="xs:string" minOccurs="0"/>
      <xs:element name="creationDate" type="xs:dateTime"
                  minOccurs="0"/>
      <xs:element name="creator" type="xs:string" minOccurs="0"/>
      <xs:element name="modifyDate" type="xs:dateTime" minOccurs="0"/>
        <xs:element name="modifier" type="xs:string" minOccurs="0"/>
        <xs:element name="assignee" type="xs:string" minOccurs="0"/>
        <xs:element name="status" minOccurs="0">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="active"/>
              <xs:enumeration value="completed"/>
            </xs:restriction>
          </xs:simpleType>
      </xs:element>
```

```
      <xs:element name="expired" type="xs:boolean" minOccurs="0"/>
      <xs:element name="expirationDate" type="xs:dateTime"
                  minOccurs="0"/>
      <xs:element name="duration" type="xs:duration" minOccurs="0"/>
      <xs:element name="priority" type="xs:int" minOccurs="0"/>
      <xs:element name="template" type="xs:string" minOccurs="0"/>
      <xs:element name="customKey" type="xs:string" minOccurs="0"/>
      <xs:element name="conclusion" type="xs:string" minOccurs="0"/>
      <xs:element name="attachment" type="xs:anyType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

The table below briefly describes the fields of the task document. More information and examples are provided in later sections.

Field	Description
`taskId` (`string`)	ID used to uniquely identify a task; automatically set by the TaskManager service when the task is created. See also `customKey`.
`title` (`string`)	Usually used when a user is presented with a list of the tasks he or she needs to complete — for example, `'Approval for order #223'`.
`creationDate` (`dateTime`)	Automatically set by the TaskManager service when the task is created.
`creator` (`string`)	ID of the application, system, or (sometimes) user initiating the task; usually used by the GUI to partition the complete worklist into categories. When the BPEL Designer is used to integrate the TaskManager service into a process, it sets this value to the name of the process initiating the task.
`modifyDate` (`dateTime`)	Managed by the TaskManager service; defines when the task was last modified.
`modifier` (`string`)	ID of the user or role performing the task update or completion operation. The semantics of the ID are owned by the application.
`assignee` (`string`)	ID of the user, role, or group responsible for completing the task. The semantics of the ID are opaque to both the Oracle BPEL Process Manager and the TaskManager service: the BPEL process sets the assignee ID, and the UI queries the list of tasks, passing in the ID.

Field	Description
`status` (`active` | `completed`)	Managed by the TaskManager service; equal to `active` or `completed`.
`expired` (`boolean`)	Managed by the TaskManager service; indicates whether the task has expired or not.
`expirationDate` (`dateTime`)	Optional; Defines when the task should expire. See also `duration`.
`duration` (`duration`)	Optional; the duration after which the task should expire. When both an expirationDate and a duration are provided, the expirationDate prevails.
`priority` (`int`)	Optional; an integer marking the priority of the task. The semantics are left to the BPEL process and GUI application.
`customKey` (`string`)	Optional; an application-specific key. The BPEL process and GUI application can use either `taskId` or `customKey` when looking up a specific task.
`conclusion` (`string`)	Optional; An application-specific field used to tell the BPEL process how the task was completed — for example, `'Approved'`, `'Refused'`, or `'Canceled'`. A common pattern in the BPEL process is that the step after the completion of the task is a `<switch>` activity that keys off the `conclusion` field. Note that this kind of information can also be passed via the `attachment` field.
`attachment` (`anyType`)	Optional; application-specific data of any type, for any purpose the application desires.

Note: Correlation and callback address information is not in the taskMessage but instead is in a WS-Addressing header.

Integrating the TaskManager Service into a BPEL Process

Now that you have a better understanding of the interface of the TaskManager service, let's see how that service can be integrated into a BPEL process. The general steps for integrating the TaskManager service within a BPEL process are:

1 Define a partnerLink for the TaskManager service.

2 Declare and initialize the task document.

3 Invoke the `initiateTask` operation of the TaskManager service just like any other standard Web service, using an `<invoke>` activity.

4 Wait for the `onTaskResult` callback from the TaskManager service, using a `<receive>` activity.

5 Read the updated task document from the callback message.

Add a TaskManager PartnerLink

You will need a partnerLink in your process to indicate that you will be calling the TaskManager service from your BPEL process (just as for any other Web service). The WSDL file for the TaskManager service can be found on your local Oracle BPEL Process Manager installation at this location:

http://localhost:9700/orabpel/default/TaskManager/TaskManager?wsdl

The outcome of adding the partnerLink will be as illustrated in the TaskSample process in `C:\orabpel\samples\tutorials\110.UserTasks`. The deployment descriptor `C:\orabpel\samples\tutorials\110.UserTasks\bpel.xml` references the TaskManager service WSDL file, as shown below.

```
<properties id="reviewManager">
   <property name="wsdlLocation">
     http://localhost:9700/orabpel/default/TaskManager/TaskManager?wsdl
   </property>
</properties>
```

The BPEL process definition at `C:\orabpel\samples\tutorials\110.UserTasks\TaskSample.bpel` uses this property to define a partnerLink for the TaskManager.

```
xmlns:task="http://services.oracle.com/bpel/task"
.
<partnerLink name="reviewManager"
             partnerLinkType="task:TaskManager"
             partnerRole="TaskManager"
             myRole="TaskManagerRequester"/>
```

Declare and Initialize the Task Document

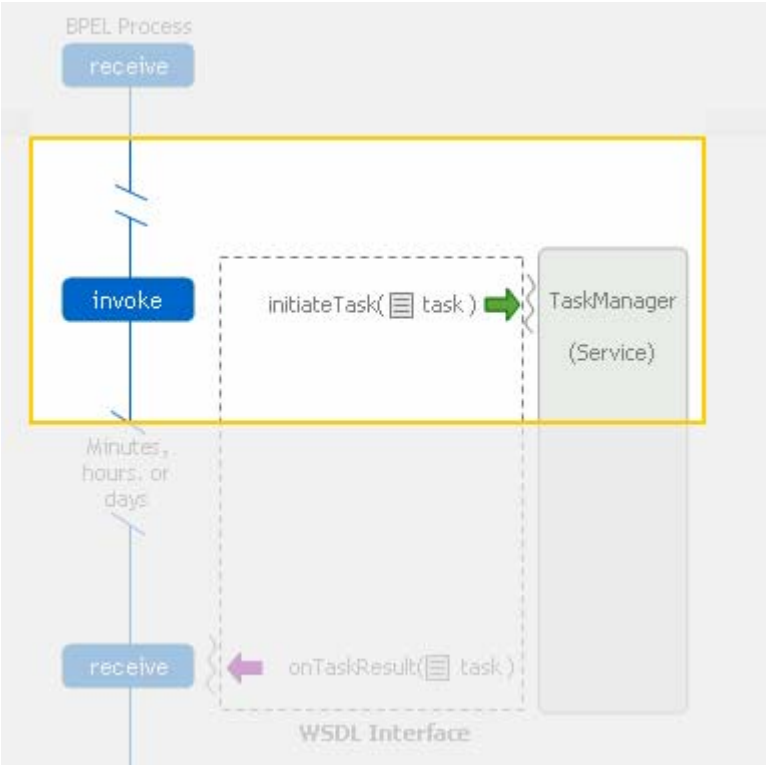
Before the TaskManager service can be invoked, a taskMessage must be constructed. The following BPEL code from `TaskSample.bpel` shows how to construct this message using the BPEL `<assign>` activity.

```
<scope name="review" variableAccessSerializable="no">
   <variables>
     <variable name="reviewTask"
               element="task:task"/>
    ...
   </variables>

   <sequence>
      ...
   <assign name="configureTask">
      <!-- Assign 'title' in task document -->
      <copy>
         <from variable="input" part="payload"
               query="/stockReviewSheet/symbol"/>
         <to variable="reviewTask" query="/task/title"/>
      </copy>
      <!-- Assign 'assignee' in task document -->
      <copy>
         <from expression="string('jsmith@finance.com')"/>
         <to variable="reviewTask" query="/task/assignee"/>
      </copy>

      <!-- ... See the full source for the other field settings -->

      <!-- Assign 'attachment' in task document -->
      <copy>
         <from variable="input" part="payload"/>
         <to variable="reviewTask" query="/task/attachment"/>
      </copy>
   </assign>
```

Note, of course, that these fields can be populated statically or using the same methods of XML data manipulation described in Tutorial 3, “Manipulating XML Documents in BPEL.”

Initiate the Task

The next step in your BPEL process is to initiate the TaskManager by invoking its `initiateTask` operation, passing the data defined above. Specifically, you will pass a taskMessage that you set up as a wrapper around the task document.



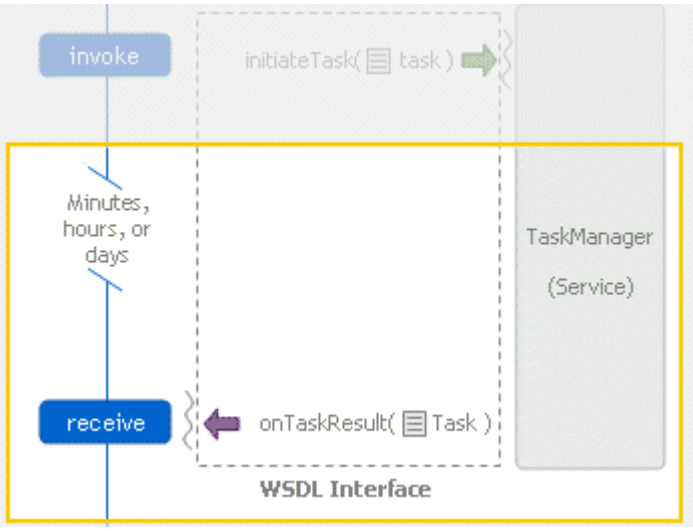

```
<scope name="reviewUserInteraction" variableAccessSerializable="no">
   <variables>
      <variable name="taskRequest" messageType="task:taskMessage"/>
         ...
   </variables>
   <sequence>
      <!-- Assign task document to taskMessage -->
      <assign name="setPayload">
         <copy>
            <from variable="reviewTask"/>
            <to variable="taskRequest" part="payload"/>
         </copy>
      </assign>

      <!-- Initiate task -->
      <invoke name="initiateTask"
              partnerLink="review"
              portType="task:TaskManager"
              operation="initiateTask"
              inputVariable="taskRequest"/>

      ...
   </sequence>
</scope>
```

At this point the task will be created and assigned to the assignee specified in the task document and will be available via the Worklist Java API.

Wait for the Task to Complete

Typically the next step in the BPEL process is to wait for the task to be completed via a `<receive>` activity; however, any arbitrary code can be executed in the interim. One

example of a common action to take in the interim is to send out some sort of notification to the user who has been assigned the task. An example of how to do email notifications for initiated tasks is described later in this document.



In any case, the BPEL code will wait for the TaskManager service to call it back. The callback passes back an updated taskMessage, indicating that the task has been completed (or has expired).

```
<variable name=" taskResponse"
   messageType="task:taskMessage"/>
...

<!--  Receive the outcome of the task -->
<receive name="receiveTaskResult"
         partnerLink="review"
         portType="task:TaskManagerCallback"
         operation="onTaskResult"
         variable="taskResponse"/>

<!-- Read task document from taskMessage -->
<assign name="readPayload">
   <copy>
      <from variable="taskResponse" part="payload"/>
      <to variable="reviewTask"/>
   </copy>
</assign>
```

The `<receive>` activity shown above will not complete until a callback is received from the TaskManager service. As with all asynchronous activities, the BPEL process will be "dehydrated" reliably during this time, and "rehydrated" and executed when the task completion (or expiration) event occurs.

The type for the TaskManager service response data is the same `taskMessage` message type that was used for the initiate message. However, since the type for the `attachment` field is an XML Schema `anyType` and is application-defined, the attachment data returned can be any type and specifically does not need to be the same type as the initiate message attachment.

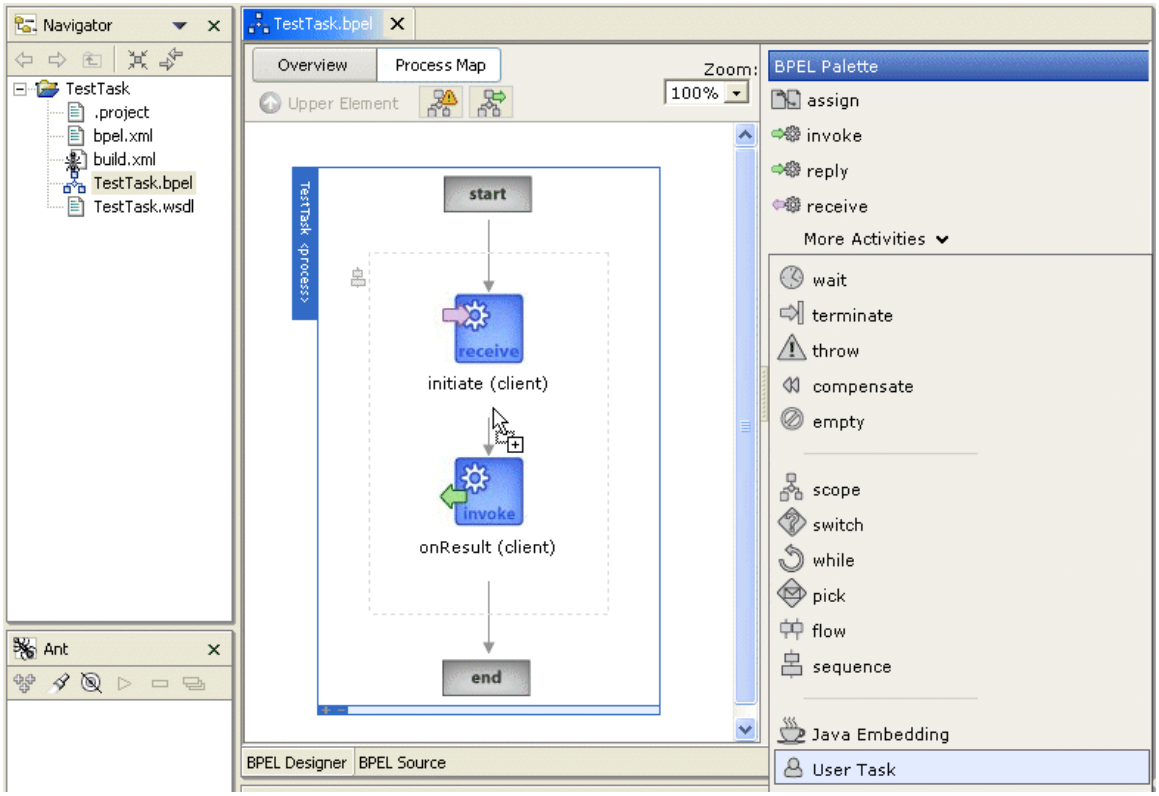
Typically, the `conclusion` field will contain information that tells the BPEL process how the task was completed (for example, `'Approved'`, `'Rejected'`, or `'Canceled'`). This kind of information could be passed via the attachment as well; it is up to the programmer to pass task data in the preferred manner.

Using the BPEL Designer to Integrate the TaskManager Service

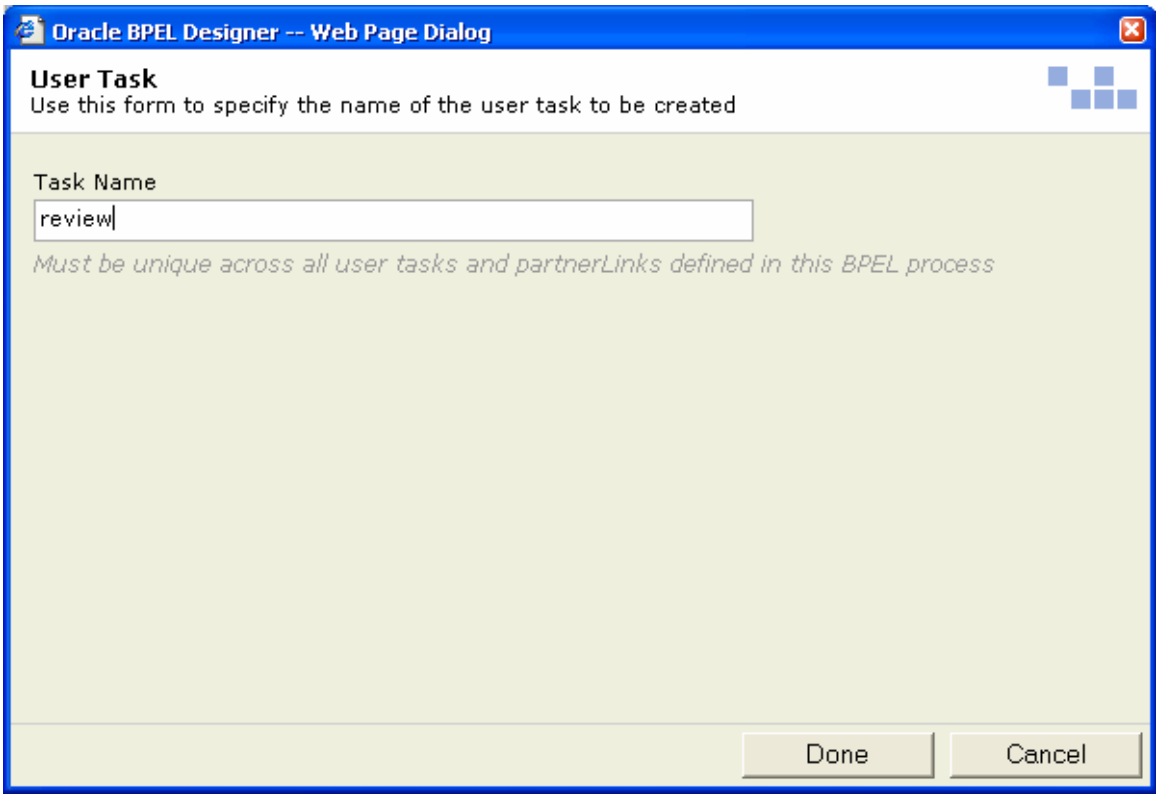
This section will review how to integrate the TaskManager service into a BPEL process using the Oracle BPEL Designer (based on version 0.6 of the Designer). It assumes you are already familiar with the basics of the Designer for creating an asynchronous process, as discussed in Tutorial 2, "Developing a BPEL Credit Flow Process."

The Designer simplifies matters considerably for you, generating much of the code described earlier in this document, by enabling you to simply drag a user task from the BPEL Palette into your process. The following steps illustrate adding a user task named `review` to a new asynchronous BPEL process (called TestTask) created in the Designer; in actual practice, of course, your starting point will be a process that you are developing.

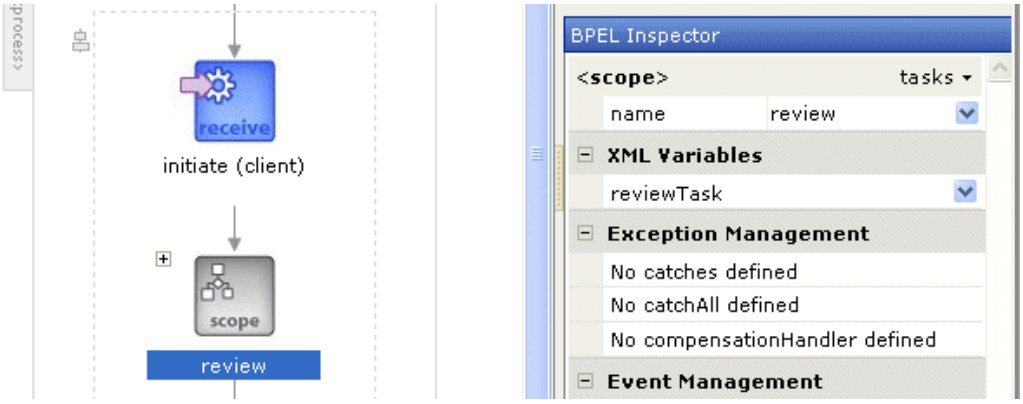
1 In the Process Map view of the BPEL file, drag a user task from the BPEL Palette (specifically, the last item in the **More Activities** list) to the transition arrow between the `initiate (client) <receive>` activity and the `onResult (client) <invoke>` callback element.



2 In the User Task dialog box that appears, enter `review` as the task name and click **Done**.

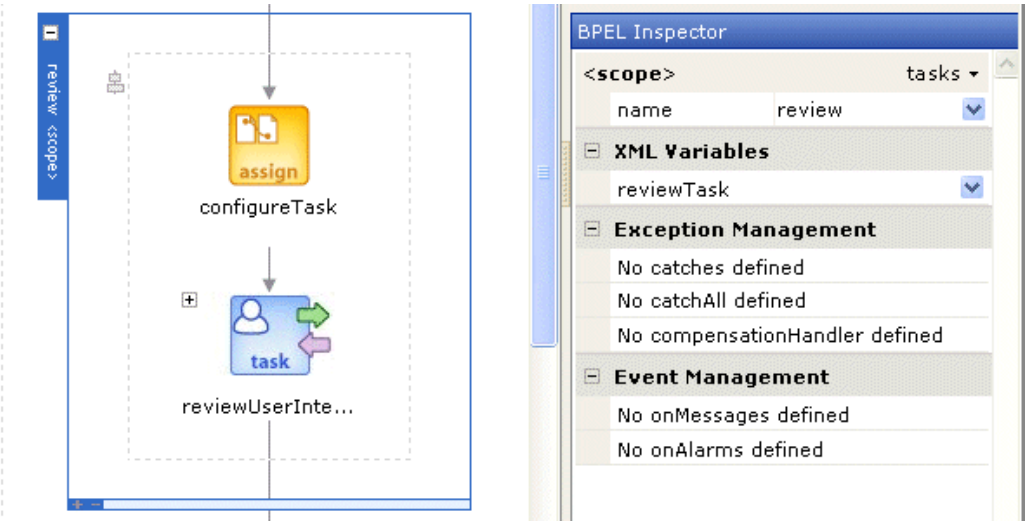


A `<scope>` activity having `review` as its `name` attribute is created for the task. As indicated by the highlighting of its name in the process map, this scope is the currently selected element, so the BPEL Inspector displays information about it — mainly, that it defines an XML variable named `reviewTask` (which is the task document).



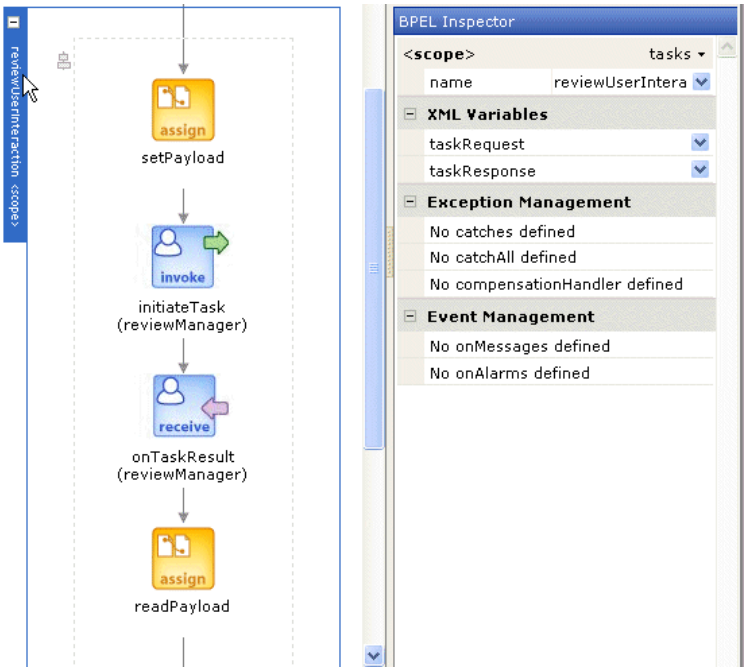
Note: All names beginning with `review` in this example will in general begin with whatever task name you specify in the User Task dialog.

3 Expand the newly created scope by clicking the "+" icon to the left of it in the process map. Within the expanded scope you will see an `<assign>` activity (named `configureTask`) and a task that can be expanded further.

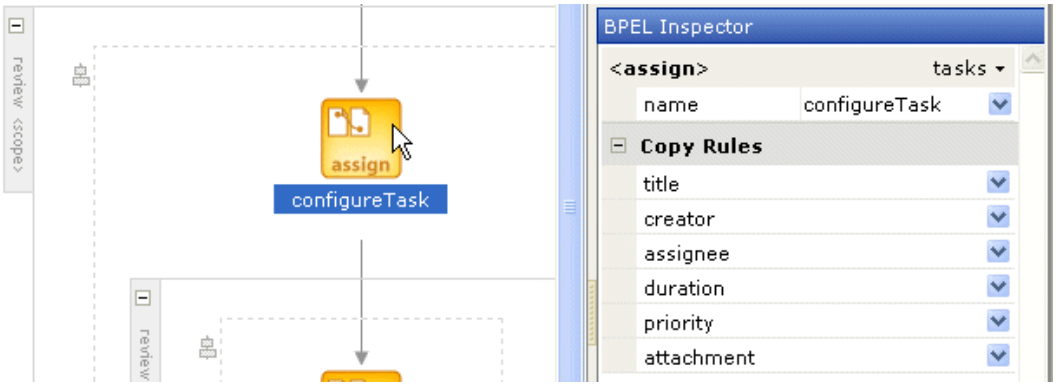


The `configureTask` assignment initializes the task document; however, before drilling down to that level, you will do one more expansion, to get an overview of the process.

4 Expand the task (`reviewUserInteraction`) by clicking its "+" icon and notice that it expands into a scope containing the `<invoke>` and `<receive>` activities for the task, along with two `<assign>` activities. Click the label along the left edge of this scope to select it so you can see what the Inspector displays for it.

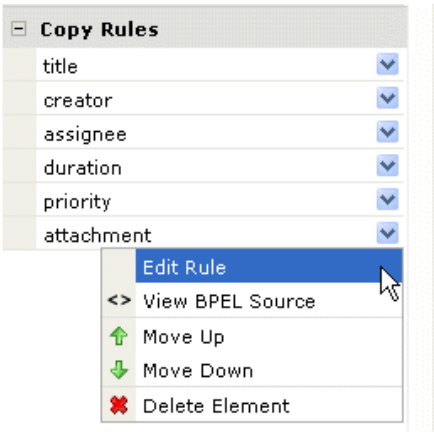


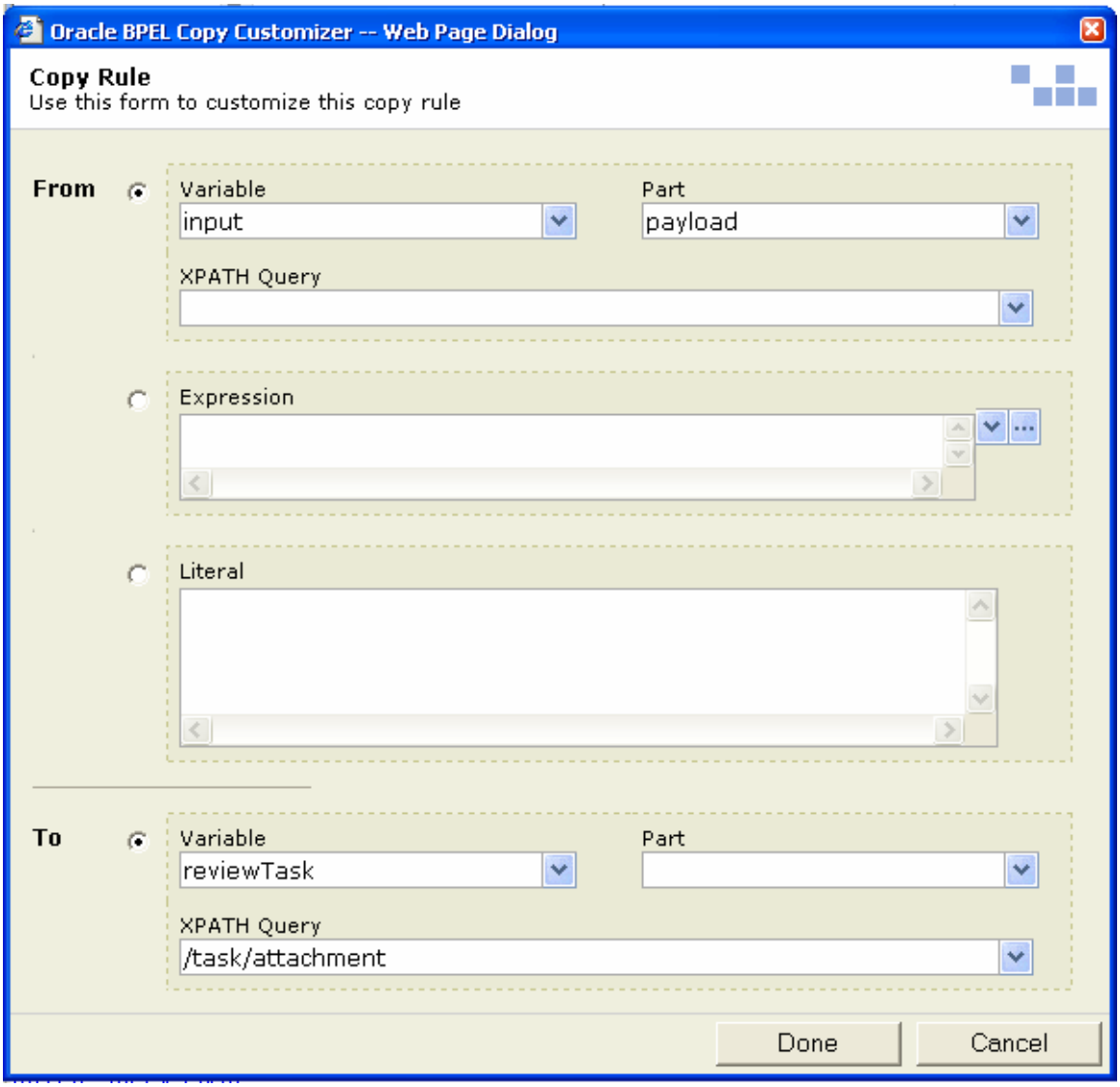
5 Scroll back in the process map if necessary to the outer scope (named `review`) and click the `configureTask` assignment within it. Listed under Copy Rules in the BPEL Inspector are the fields of the `reviewTask` task document.



You can view the Copy Rule form for each field either by selecting **Edit Rule** in its drop-down list or by clicking the field name (which is a link). For every field except the last one, `attachment`, the Copy Rule form will show an assignment being made to that field. In actual practice, you would change the values being assigned to the fields `title` through `priority` as appropriate, and optionally pass custom data to the task via the `attachment` field (as in the next step).

6 To pass data to the task, click **Edit Rule** in the drop-down list for the `attachment` field in the Inspector (or simply click `attachment`) and fill in the **From** part of the Copy Rule form accordingly — to pass, for example, the `payload` part of the `input` variable (as done earlier under "Declare and initialize the task document"). Note that if you do not want to pass anything in the `attachment` field, you *must* click **Delete Element** in the field's drop-down list (or the process will not compile because of the incomplete assignment).





7 Click **Done** to complete the configuration of the attachment copy rule.

8 Click the next `<assign>` activity in the process map: `setPayload`, inside the `reviewUserInteraction` scope. You will see that only `payload` is listed under Copy Rules and, if you look at the Copy Rule form, that `reviewTask` (the task document, from the outer scope) is assigned to the `payload` part of the `taskRequest` variable as the final step before initiating the task.

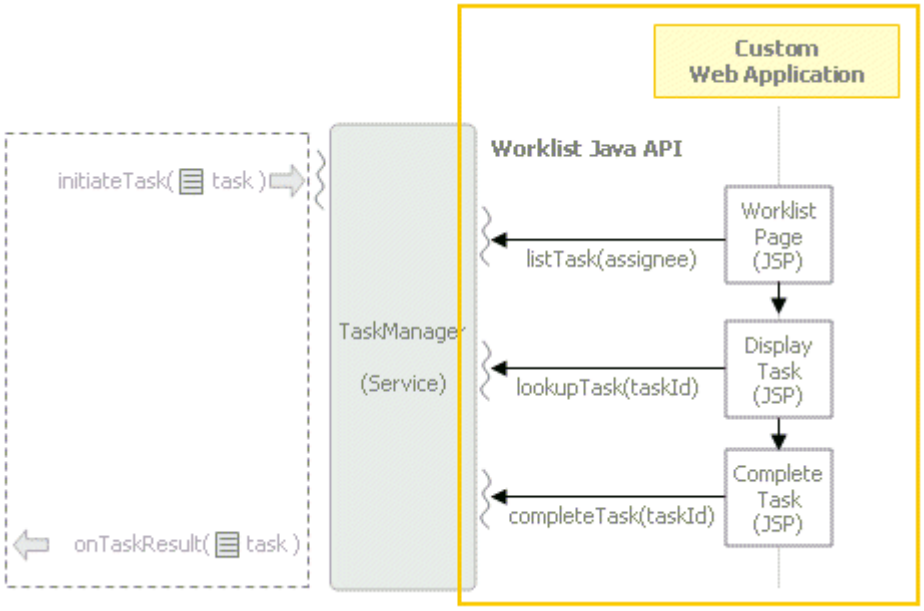
9 Click the final `<assign>` activity (`readPayload`) and explore it in the Inspector to see that this is where the task data returned in a taskMessage (that is, in the `payload` part of the `taskResponse` variable) is read and stored in the variable `reviewTask`.

Note that while the above may seem complex at first, it is due to the fact that the TaskManager service is a complex service. The Designer User Task Palette entry is merely a pre-configured template that encapsulates the normal patterns for use of the TaskManager service in a BPEL flow. However, the Designer will continue to improve

as a means of integrating the TaskManager service into a BPEL process — and, of course, you always have the option of working directly in the BPEL source code.

Creating the User Interface for the Task

On the other side of a TaskManager service is usually a user interface through which the tasks assigned to a user or role are displayed, task data is viewed and updated, and tasks are completed.



The typical steps for accessing task information from the user side are:

1 A GUI uses the Java Worklist API to list the tasks assigned to a user or role.

2 The user selects a task to see detail information for it.

3 The user updates any editable data associated with the task and saves or (eventually) completes the task.

List the Assigned Tasks

Using the Java Worklist API to the TaskManager service from the client perspective involves the following two high-level steps. (Note that all the code shown in this section is taken from `C:\orabpel\samples\tutorials\110.UserTasks\TaskSampleUI\listTasks.jsp`.)

1 Use the `com.oracle.bpel.client.Locator` class to get back a worklist service handle (an IWorklistService).

```
<%@page import="com.oracle.bpel.client.Locator" %>
<%@page import="com.oracle.bpel.services.task.IWorklistService" %>
   ...

   // Obtain a reference to BPEL domain 'default' using password 'bpel'
   Locator locator=new Locator("default", "bpel");

   IWorklistService worklist =
     (IWorklistService)
        locator.lookupService(IWorklistService.SERVICE_NAME);
```

2 Use the `com.oracle.bpel.services.task.IWorklistService` interface to fetch `com.oracle.bpel.services.task.ITask` objects, manipulate them, complete them, and so on.

```
<%@page import="com.oracle.bpel.services.task.ITask" %>
   ...
   ITask[] tasks=
       worklist.listTasksByAssignee("jsmith@finance.com");
   ...
```

You can fetch tasks by other criteria as well. For example, you can look up a task by its unique taskId assigned by the TaskManager service or based on a specified `creator` attribute (which is programmer-defined). You can also build an arbitrary search criterion using `the com.oracle.bpel.client.util.WhereCondition` utility class — for example, to fetch all the expired tasks assigned to a particular user. For more information, including full documentation on the interfaces for the classes used in the code in this section, see the Oracle BPEL Process Manager API Javadocs (located at `C:\orabpel\docs\apidocs\index.html` for a default Oracle BPEL Process Manager installation).

The code below shows how to iterate through the list of tasks in the JSP page and, for each task, display some basic descriptive information and a link that brings up a detail page (`displayTask.jsp`) for the task.

```
<h1>My Custom Task List Page</h1>
<table border="1">
<tr>
    <th>Title</th>
    <th>Due Date</th>
    <th>Priority</th>
</tr>
<%
    // Iterate through the list of tasks
    for (int i=0; i < tasks.length; i ++)
    {
        ITask thisTask=tasks[i];

        // We are interested in tasks assigned to "jsmith@finance.com"
        // and generated by the TaskSample BPEL process. Often the BPEL
        // process initiating a task will assign its name to the
        // creator field  of the task document. This is at least the
        // case with the TaskSample BPEL process.
        if (! "TaskSample".equals(thisTask.getCreator()))
```

```
        {
            // This task has been generated by another BPEL process
            // and should therefore not be in the list.
            continue;
            }

            // Read the title assigned to the task. The title is to
            // a task what a subject would be to an email.
            String title=thisTask.getTitle();

            // Get the unique ID/key associated with this task.
            // This is passed from page to page to identify the
            // task the user is reviewing and completing.
            String taskId=thisTask.getTaskId();

            // Read the expiration date assigned to the task.
            Date expiration=null;

        if (thisTask.getExpirationDate() != null)
            expiration=thisTask.getExpirationDate().getTime();

        // Read priority associated with the task
        int priority=thisTask.getPriority();
%>
        <tr>
            <td>
            <a href="displayTask.jsp?taskId=<%= taskId %>">
               <%= title %></a>
            </td>
            <td>
            <%= expiration %>
            </td>
            <td>
            <%= priority %>
            </td>
        </tr>
<%
    }
%>
</table>
(extracted from TaskSampleUI/listTasks.jsp)
```

Display the Payload Data/Detail for a Task

The `displayTask.jsp` page invoked from the hyperlink in the code above gets passed a taskId (again, a unique string identifier generated by the TaskManager service) and uses the same `Locator` and `IWorklistService` objects as above to fetch the task associated with that taskId. Then, to manipulate the payload data associated with a task, it uses the façade capability, which provides a thin Java interface on top of arbitrary XML data. Note that façades are not at all like JAXB-style bindings because they do not attempt to fully map XML to Java; they merely provide a Java envelope for XML data which is then accessed via JavaBeans-style getter and setter functions. This enables façades to avoid all of the problems associated with JAXB bindings as schemas get complex.

The façade construct is described in more detail in the Oracle BPEL Process Manager tutorial on invoking Java from BPEL; however, at a very high level, the basic steps for working with the XML façade are as follows:

1 In your `build.xml` script, use the `schemac` Ant task (or command-line tool) to generate façade classes for XML Schema types. For example:

```
<schemac input="${basedir}/TaskSample.wsdl"
         out="${basedir}/TaskSampleUI/WEB-INF/classes"/>
```

This command generates the façade classes for the XML Schema types described in `TaskSample.wsdl`, namely:

```
<element name="stockReviewSheet" type="finance:StockReviewSheetType" />

<complexType name="StockReviewSheetType">
    <sequence>
       <element name="symbol"       type="string" />
       <element name="targetPrice"  type="float"  />
       <element name="currentPrice" type="float"  />
       <element name="action"       type="string" />
       <element name="quantity"     type="int"    />
    </sequence>
</complexType>
```

The façade classes generated include a factory for creating new instances of the `stockReviewSheet` element and then getter and setter methods for manipulating the data fields within the instances. If you want to review the exact interfaces exposed by the generated classes, you can either view the generated Java classes or create Javadocs for them by using the `-j` option to the `schemac` command.

2 Within your code, you can use the generated classes to manipulate instances of XML data types through a simple Java interface.

The code below (from `C:\orabpel\samples\tutorials\110.UserTasks\TaskSampleUI\displayTask.jsp`) uses the `Locator` and `IWorklistService` objects to fetch a particular task by taskId. It then gets the payload of the task, which is a W3C DOM `Element` type, and uses that and the façade `StockReviewSheetFactory` class to instantiate a `StockReviewSheet` façade instance. Finally, the façade class getter and setter methods are used to fetch appropriate data fields from the payload and display them in the JSP page.

```
...
<%@ page import="com.otn.samples.finance.StockReviewSheet" %>
<%@ page import="com.otn.samples.finance.StockReviewSheetFactory" %>...

Locator locator=new Locator("default", "bpel");

IWorklistService worklist=(IWorklistService)
    locator.lookupService(IWorklistService.SERVICE_NAME);

ITask task=worklist.lookupTask(taskId);

// Get a reference to the XML StockReviewSheet document attached to the
// task. If you look at the implementation of the TaskSample BPEL
// process, you will notice that a StockReviewSheet element is assigned
// to the task as an attachment.
Element rsElement=(Element) task.getAttachment();

StockReviewSheet srs =
   StockReviewSheetFactory.createFacade(rsElement);

// Use the friendly bean interface of the XML facade to access the data
// contained in the XML attachment.
String symbol=srs.getSymbol();
float targetPrice=srs.getTargetPrice();
float currentPrice=srs.getCurrentPrice();
...

<tr>
    <td>Symbol</td>
    <td><input type="text" name="Symbol"
               value="<%= symbol %>"/></td>
    <td>(String)</td>
</tr>
...
(extracted from TaskSampleUI/displayTask.jsp)
```

The `displayTask` JSP page provides an HTML form that enables the user to fill in the `action` and `quantity` fields for the stock review sheet, as well as to edit any of its other fields. When the form is submitted, it passes these values, along with the taskId, to `completeTask.jsp`.

Update the Payload Data and Complete the Task

The final step is to use the generated façade classes to create the appropriate attachment data element (the task document) to be returned to the task and then pass it to the TaskManager service along with an indication that the task has been completed.

In this example, this is done in `completeTask.jsp`, which creates a new instance of the `StockReviewSheet` element and fills it in with the form data passed to it. It then looks up the task by taskId, as before. What is new about this code (from `C:\orabpel\samples\tutorials\110.UserTasks\TaskSampleUI\completeTask.jsp`) is shown below. It sets some fields of the `task` object (including the updated attachment) and then informs the worklist service that the task has been completed.

```
StockReviewSheet srs=StockReviewSheetFactory.createFacade();

// Populate symbol from data submitted as part of the post
srs.setSymbol(request.getParameter("symbol"));
...

ITask task=worklist.lookupTask(taskId);

task.setAttachment(srs.getRootElement());

// Conclusion is user-defined - here it is "buy" or "sell"
task.setConclusion(strAction);

worklist.completeTask(task);

out.println("This task has been successfully completed.");
```

At this point, the TaskManager service will call back asynchronously to the process that is waiting for this task, passing it the updated task attributes and attachment data.

Deploying and Testing the TaskSample Process

This section describes how to compile, deploy, and test the TaskSample process, using the BPEL Console to initiate the process and the UI described above for the stock review sheet review task.

To compile and deploy the sample process and UI JSP pages:

1 Make sure your Oracle BPEL Process Manager is started.

2 Open up a command prompt.

3 **cd C:\orabpel\samples\tutorials\110.UserTasks**

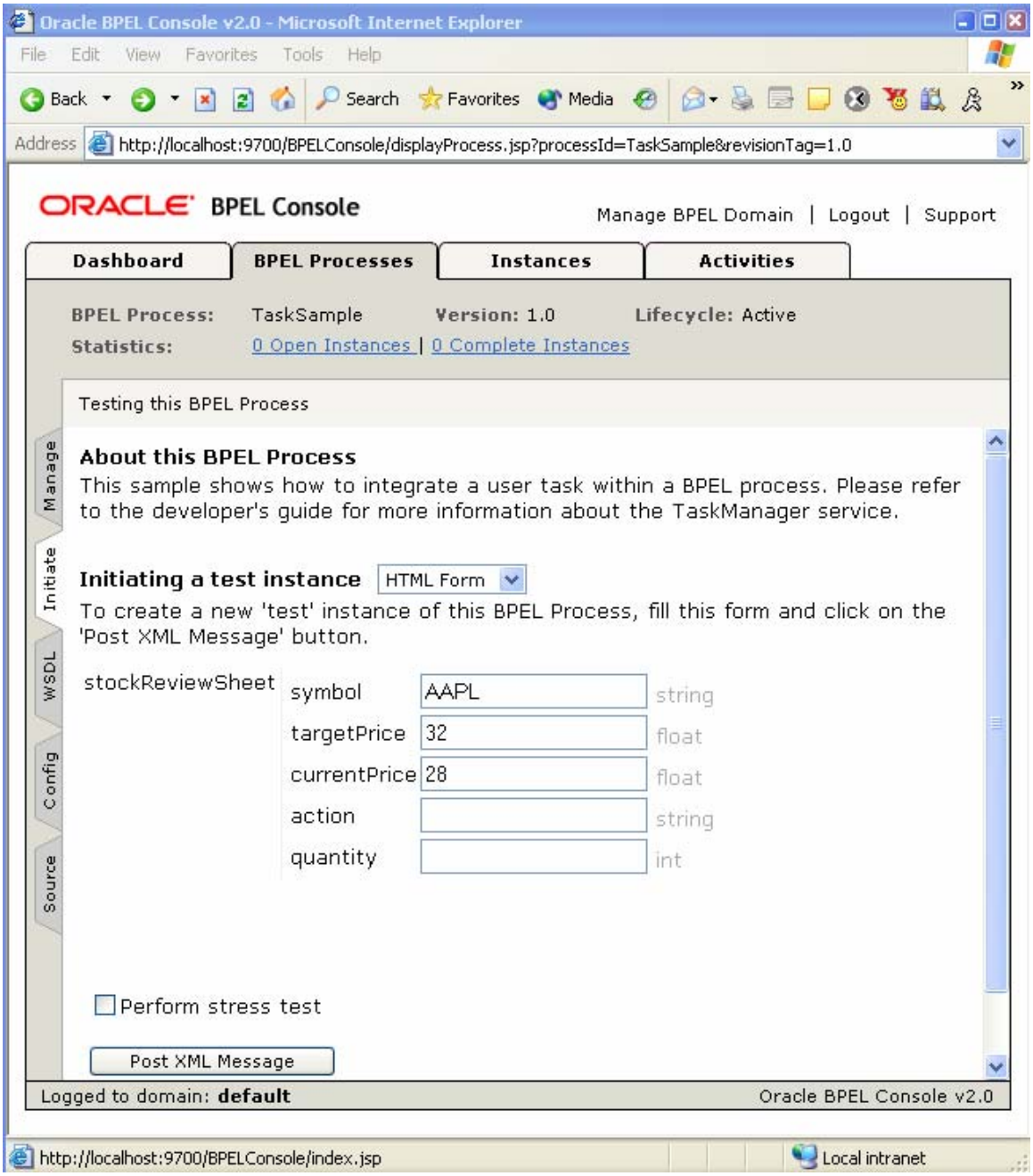
4 **obant**

To test:

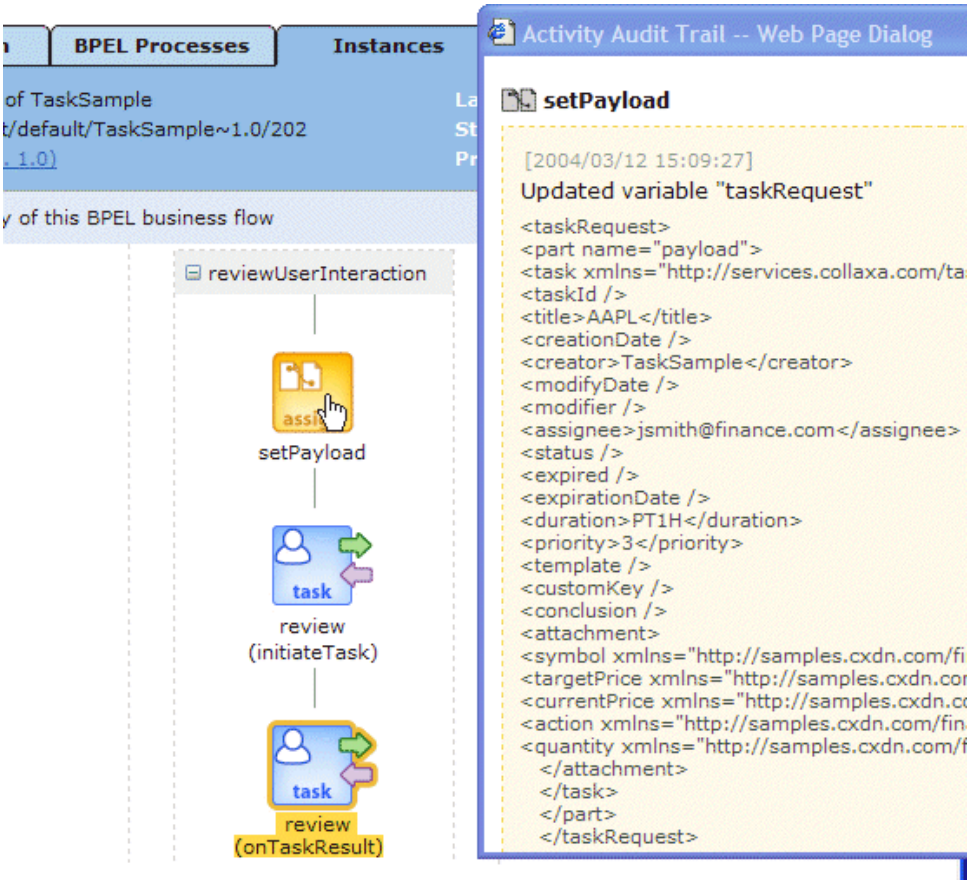
5 Connect to the BPEL Console at http://localhost:9700/BPELConsole and log in to the default domain if necessary.

6 Select the TaskSample process in the list of deployed BPEL processes.

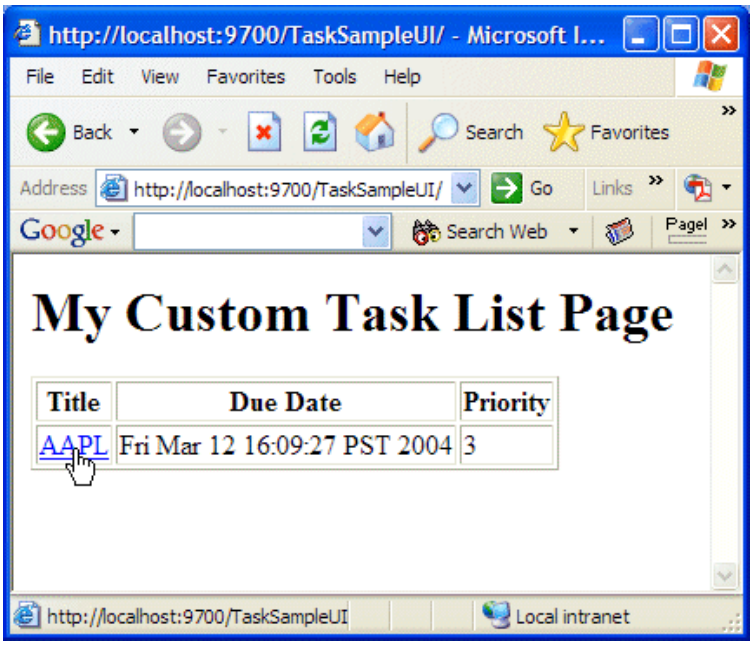
7 Enter any values for the `stockReviewSheet` `symbol`, `targetPrice`, and `currentPrice` fields, and click **Post XML Message**. (You can leave the `action` and `quantity` fields blank.)



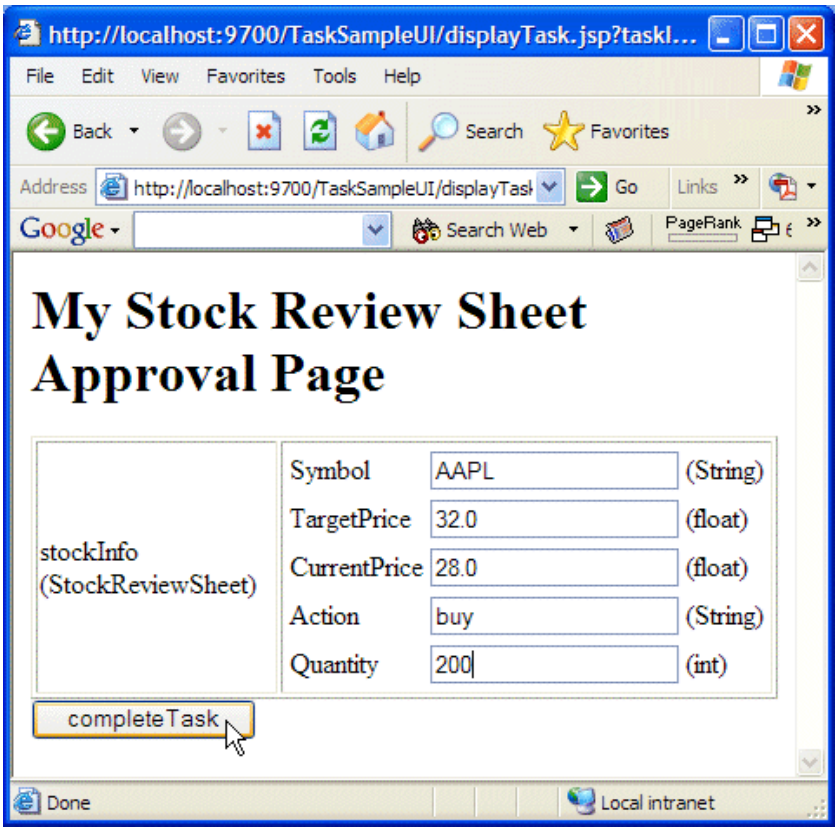
8 Select the **Visual Flow** link to see the visual audit trail. In the visual audit trail you can see that the process has initiated an instance of the TaskManager service and assigned a task to jsmith@finance.com to review the data and supply an action and a quantity.



9 Bring up the TaskSample UI with the URL http://localhost:9700/TaskSampleUI.

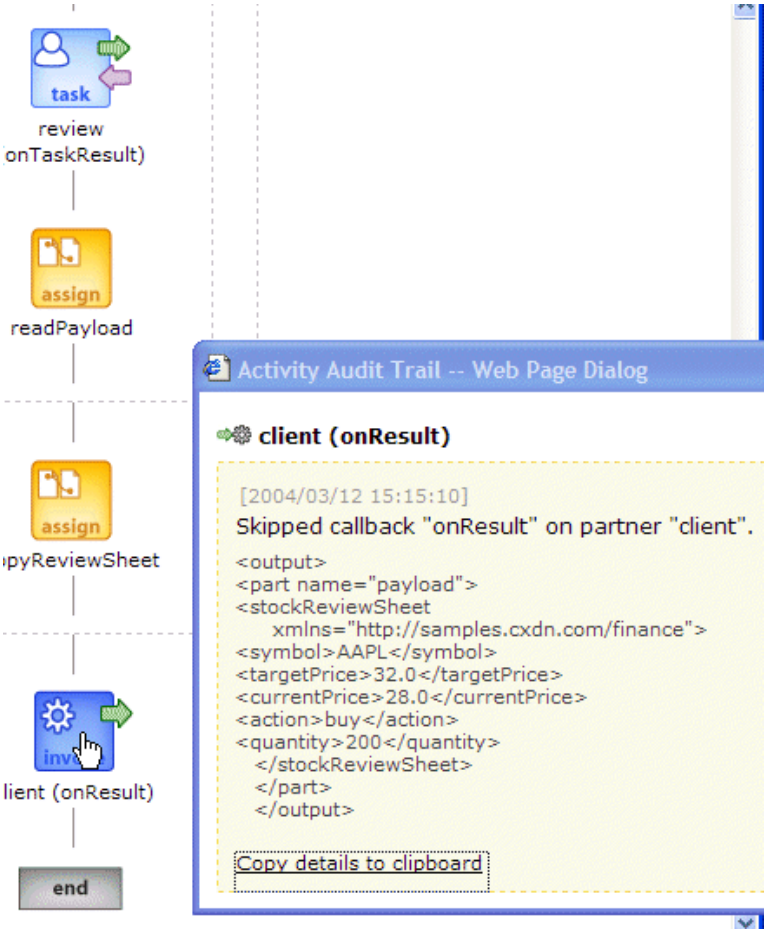


10 Select the task title to see the detail information for it. Fill in the Action and a Quantity and click **completeTask**.



This will complete the task and notify the waiting BPEL process.

11 Refresh the audit trail for the TaskSample instance in the BPEL Console and notice that the task has completed, passing back the updated data, and the flow has completed, returning the complete `stockReviewSheet` data to the caller.



Additional Capabilities

This final section will look at some additional capabilities related to working with the TaskManager service: expiration/timeouts; notification to the assignee; task reassignment; and task assignment to groups (and role resolution).

Enabling Expiration/Timeouts for Tasks

The TaskManager service has a built-in expiration/timeout capability. You can specify the timeout period for a task as a period of time (`duration` field) or a specific point in time (`expirationDate` field). In either case, when the specified time is reached and the task has not yet been completed, an expiration event is sent as a BPEL event to the process that is waiting for completion of that TaskManager service. The waiting BPEL process would then use a BPEL eventHandler, and at that point could take any desired action (including sending a reminder email, reassigning the task to someone else, canceling the task, and so on).

The `duration` and `expirationDate` fields are specified as having the XML Schema types `duration` and `dateTime`, respectively. For example, a duration of one hour is specified as `'PT1H'` (meaning a **P**eriod of **T**ime of **1 H**our). In addition to "H" for hours, you can also use "M" for minutes, "S" for seconds, and so on. For more information on the format of the `duration` and `dateTime` data types, see the XML Schema specification

(http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration and http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#dateTime).

First, in the BPEL source you would specify a duration or expiration date, as in the following:

```
<!-- Assign 'duration' in task document -->
<copy>
    <from expression="string('PT5M')"/>
    <to variable="reviewTask" query="/task/duration"/>
</copy>
```

Next, after initiating the task, you would:

1 Wrap the TaskManager service `<receive>` activity in a scope.

2 Add an eventHandler to the scope to handle `onTaskExpired` events.

3 Within that eventHandler, do whatever you like in response to the expiration event. The code below reassigns the task to someone else and sets a new expiration time of one more hour.

```
<invoke name="initiateTask"
        partnerLink="review"
        portType="task:TaskManager"
        operation="initiateTask"
        inputVariable="taskRequest"/>

<!-- Receive the outcome of the task -->
<scope name="reviewTask">
  <eventHandlers>
      <onMessage partnerLink="review"
                 portType="task:TaskManagerCallback"
                 operation="onTaskExpired"
                 variable="taskRequest">
        <sequence>
            <assign name="reassignTask">

                <!-- Assign NEW 'assignee' in task document -->
                <copy>
                    <from expression=
                              "string('director@oracle.com')"/>
                    <to variable="taskRequest"
                        part="payload"
                        query="/task/assignee"/>
                </copy>

                <!-- Assign NEW 'duration' in task document -->
                <copy>
                    <from expression="string('PT1H')"/>
                    <to variable="taskRequest"
                        part="payload"
                        query="/task/duration"/>
                </copy>
```

```
                <!-- Assign NEW 'status' in task document -->
                <copy>
                    <from expression="string('Escalated')"/>
                    <to variable="taskRequest"
                        part="payload"
                        query="/task/status"/>
                </copy>
            </assign>
            <invoke name="reassign"
                    partnerLink="review"
                    portType="task:TaskManager"
                    operation="updateTask"
                    inputVariable="taskRequest"/>
        </sequence>
      </onMessage>
  </eventHandlers>
  <receive name="receiveTaskResult"
           partnerLink="review"
           portType="task:TaskManagerCallback"
           operation="onTaskResult"
           variable="taskResponse"/>
</scope>
```

If you want to run the entire example, note that the code example above comes from the TimeOffRequestFlow sample available from http://otn.oracle.com/bpel.

Sending Notifications

It is common to send some sort of notification message to a user when a task is assigned to that user (or when the timeout duration is reached, when the task is reassigned, or in other situations). This notification can be done in BPEL at the same time that the TaskManager service is initiated; therefore, anything that can be done in BPEL (invoking a Web service, sending an email message or a JMS message, executing some Java code, and so on) can be done to notify a user of a task-related event.

The TimeOffRequestFlow sample includes a code example of using the Oracle BPEL Process Manager MailService to send an email notification to a user when a time-off request approval task has been assigned to that user. In this case, the email includes an XML document attachment describing the time-off request and contains a link to a JSP page where the request can be approved or rejected.

Reassigning Tasks

Reassigning a task is as simple as changing the `assignee` field in the task document and then invoking the `updateTask` operation on the TaskManager service. A code example of this is shown earlier in the section “Enabling expiration/timeouts for tasks.” If the application requires that once a task is reassigned it cannot be completed by the original assignee, then the UI that completes a task should first fetch the task document and check that the assignee has not changed before completing a task.

Assigning Tasks to Groups and Resolving Roles

The TaskManager service has a very simple construct for the `assignee` attribute of a task: it is a string identifier that is not interpreted by the TaskManager service to have any special meaning. Although simple, this construct is surprisingly powerful and flexible. In particular, it allows for role resolution and the concept of group worklists to be implemented in a straightforward manner.

When a task is assigned to a group, typically this means that several people will be able to see the task in their worklists and, when one of them selects the task to work on, it will disappear from the worklists of the other users in the group. Additionally, it is common to set up a timeout period such that if the individual working on the task does not complete it within a specified time, it may revert back to group assignment and then reappear on the group worklists.

The most common way to implement this use case with the TaskManager service is as follows:

1 The task is assigned to the group identifier.

2 The worklist UIs for members of the group will show all tasks assigned to the group.

3 When a user selects a task to work on, it is checked to make sure it has not already been reassigned and, if not, it is reassigned to the individual's user ID.

4 Optionally, notifications and timeouts can be used effectively here.

Role resolution works in much the same way. Because special interpretation is not given to the assignee for a task, the assignee can actually be a role, and resolution can happen on the UI side. Alternatively, it is fairly simple to integrate a BPEL process with directory services such as LDAP and perform dynamic role resolution at the time the task is created.