Tutorial: Building a Worklist Application
Contents

1. Tutorial: Overview
   Tutorial Overview ................................................................. 1-2
   Steps in This Tutorial ......................................................... 1-3
   Step 1: Setting Up the Environment ........................................... 1-3
   Step 2: Modeling and Deploying the Loan Processing Task Plan ........... 1-3
   Step 3: Testing the Task Plan Using Worklist User Portal ................... 1-3
   Step 4: Managing Task Instances Using Worklist Console .................. 1-4
   Step 5: Using JPDs with Worklist .............................................. 1-4
   Conventions ........................................................................... 1-4

2. Step 1: Setting Up the Environment
   Before You Begin ................................................................... 2-1
   Create a Worklist Domain ....................................................... 2-2
   Set Up the Workshop for WebLogic Platform Design-Time Environment ........ 2-8
   Configure Users and Groups for Loan Processing ............................ 2-10
   Create Groups for the Loan Processing Task .................................. 2-11
   Create Users and Assign to Groups ............................................ 2-13
   Create a Business Calendar ....................................................... 2-14

3. Step 2: Modeling and Deploying the Loan Processing Task Plan
   Model and Deploy the Loan Processing Task Plan ........................... 3-1
   Create a New Worklist Application ............................................. 3-2
Create a New Task Plan ................................................................. 3-5
Define the Steps for the Loan Processing Task Plan ..................... 3-6
Define Actions in the Task Plan ...................................................... 3-10
Define Constructors for the Task Plan .......................................... 3-12
Validate the Task Plan ................................................................. 3-14
Deploy the Loan Processing Task Plan .......................................... 3-15

   Create the Loan Processing Task Instance ................................... 4-1
   Claim the Loan Processing Task Instance .................................. 4-3
   Reject the Loan Task Instance .................................................. 4-6

5. Step 4: Managing Task Instances Using Worklist Console
   Update the Application Using Worklist Console .......................... 5-1
   Verify Updated Application in Worklist Portal .............................. 5-5

6. Step 5: Using JPDs with Worklist
   Subscribe to Worklist Events ...................................................... 6-1
   Configure a Perform Node ......................................................... 6-4
   Verify the Worklist Event is Published ...................................... 6-6
   Use the Worklist Control ........................................................... 6-8
   Create a Worklist JPD .............................................................. 6-8
   Create a Task Control .............................................................. 6-11
   Add Task Plan Constructor to JPD ............................................. 6-13
   Validate the WorklistControl JPD ............................................ 6-15

7. Advanced Topic: Adding a Customized User Interface
   Define Web Page Mock-Up and Flow .......................................... 7-1
   Define Page Flow ................................................................. 7-3
Define Form Beans ................................................................. 7-5

................................................................. 7-6
Define Actions on the Page Flow ........................................ 7-7
Define JSP pages ................................................................. 7-7
Register the Custom UI ......................................................... 7-7
Deploy the Custom Task UI ..................................................... 7-7
Validate the Custom UI .......................................................... 7-8

 Configure users and groups .................................................... 7-8
 Create Users ................................................................. 7-8
 Create a Loan Approval Task ................................................... 7-9
 Assigned LoanOfficer Approves Loan to his Manager ................. 7-9
 Assigned LoanManager Rejects the Loan................................. 7-10
Java Process Definition (JPD) functionality available with BEA WebLogic Integration™ enables integration of diverse systems, applications, and human participants. WebLogic Integration Worklist enables people to collaborate as part of higher level business processes.

The Worklist subsystem enables human interaction with business processes. Worklist provides the capability to assign tasks to human users. Based on the assigned task, human users can perform actions on the tasks, which can trigger new task assignments to other users or system events. This process flow depends on the higher level business processes.

Some of the Worklist subsystem features include:

- Creating and assigning tasks to users
- Generating notifications of task assignments and task due events
- Tracking task history and status

This chapter provides an overview of the tutorial and explains the business scenario on which this tutorial is based. It includes the following sections:

- Tutorial Overview
- Steps in This Tutorial
- Conventions
Tutorial Overview

This tutorial provides you step-by-step instructions to create a loan approval tracking system for a fictitious financial institution, Acme Financial System (AFS), using human interaction and system integration functionality available with Worklist.

Figure 1-1 illustrates the loan approval tracking system scenario that you will create during the course of this tutorial.

Figure 1-1  Loan Approval Tracking System Process Flow

Following is the sequence of events illustrated in the preceding figure:

1. A customer submits a loan request for an amount of $10,000 with AFS. Based on the process flow, the system routes this loan application to be reviewed by Loan Officers.

2. The Worklist creates the loan task for this loan application. This task appears in the Inbox of all users who belong to the Loan Officers group.

3. One of the Loan Officers (John) logs into the system, reviews the loan request, and claims it as his task.

4. John assesses the loan request and decides to route to the group of Loan Managers for approval.
5. Loan Manager Mary logs in and claims the loan task.

6. Mary has the choice to approve or reject the loan. If she approves the loan, Worklist flags the task as complete and triggers an event, which is sent to the loan request system.

7. If Mary rejects the loan, Worklist aborts the task and rejects the loan request.

Steps in This Tutorial

This tutorial provides you with detailed instructions to:

- Set up the environment
- Create a task plan
- Test the loan processing approval system using Worklist User Portal
- Manage loan processing task instances using Worklist Console
- Use JPDs to interact with loan task instance

In the tutorial, these tasks have been structured and categorized into six steps. These are:

Step 1: Setting Up the Environment

The section details the steps required to set up the Worklist Design-Time environment (for modeling the target task plan), and the Worklist run-time environment (for running the application.) In addition, you will need to set up the users, groups, and business calendars.

Step 2: Modeling and Deploying the Loan Processing Task Plan

This section discusses how the loan processing task plan can be modeled to implement the business scenario and then be deployed on the WebLogic Integration server.

Step 3: Testing the Task Plan Using Worklist User Portal

The Worklist User Portal is provided out of the box to allow different human users to work on the task instances. A loan processing task can be created and worked on to completion by many users through the Worklist User Portal.
Step 4: Managing Task Instances Using Worklist Console

In this section, the administrator uses the Worklist Console to look at the overall statistics of the loan processing task instances. In addition, the administrator reassigns some task instances because the assigned employee has left the company.

Step 5: Using JPDs with Worklist

In this section, the loan processing requests come from an online system that triggers a JPD business process, which creates the task instance.

Conventions

This section describes the conventions used in the text and code examples of this tutorial. Table 1-1 lists the meaning and examples for different text and code conventions.

Table 1-1  Worklist Tutorial Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Bold</strong></td>
<td>Bold typeface indicates terms on which users perform actions.</td>
<td>Click <strong>Next</strong> to proceed with the Configuration Wizard.</td>
</tr>
<tr>
<td><strong>Italics</strong></td>
<td>Italic typeface indicates names of processes.</td>
<td>Because the task instance, <em>Car loan for Maggie May</em>, is assigned to the <em>loanOfficers</em> group, it will show up on the Assigned Tasks portlet of John’s Inbox.</td>
</tr>
<tr>
<td><strong>Code</strong></td>
<td>Code typeface indicates names of files, directories, steps, and group names.</td>
<td>Specify the Web project name as <em>Loan_Web</em> in the Web Project Name box.</td>
</tr>
</tbody>
</table>
Step 1: Setting Up the Environment

In this step, you will set up the Worklist design-time environment for building the loan processing approval task plan and a new Worklist application. To set up the environment, you will create a new domain, users, and groups for the Acme Financial System.

To complete the tasks in this step, go through the following sections:

- Before You Begin
- Create a Worklist Domain
- Set Up the Workshop for WebLogic Platform Design-Time Environment
- Create a New Worklist Application
- Configure Users and Groups for Loan Processing

Before You Begin

Before you begin this tutorial, ensure that you have WebLogic Platform version 9.2 with WebLogic Integration 9.2 installed on your system and define a workspace. Perform the following to start the BEA Workshop for WebLogic Platform™ IDE (Integrated Development Environment).

1. From the Start menu, click All Programs→BEA Products→Workshop for WebLogic Platform to start the BEA Workshop for WebLogic Platform IDE. This will display the Workspace Launcher dialog box.
2. For the purpose of this tutorial, create a workspace called **worklist** in the `BEA_HOME\user_projects\w4WP_workspaces\` directory. Where `BEA_HOME` is the location where you installed WebLogic Platform 9.2 (see Figure 2-1).

**Figure 2-1  Setting the Workspace**

![Workspace Launcher](image)

Create a Worklist Domain

The Worklist domain is created using the Configuration Wizard. To create the Worklist domain:

1. From the **Start menu**, click **All Programs→BEA Products→Tools→Configuration Wizard** to start the BEA WebLogic Configuration Wizard. This displays the Welcome page in the BEA WebLogic Configuration Wizard dialog box (see Figure 2-2).
2. Select **Create a new WebLogic domain** and click **Next**. This displays the Select Domain Source page in the Configuration Wizard dialog box.

As you proceed through the Configuration Wizard, several pages will appear in a sequence. You need to specify your settings on each page and click **Next** to proceed to the subsequent page. **Table 2-1** lists the pages and the options that you need to select to create the domain successfully.
3. On the Create WebLogic Domain page, specify the following values for each field and click Create (see Figure 2-3):

- **Domain name**: myworklist
- **Domain Location**: C:\bea\user_projects\domains
- **Application Location**: C:\bea\user_projects\applications
After the domain is created successfully, the Creating Domain page is displayed (see Figure 2-4).
4. Select the **Start Admin Server** check box and click **Done** to proceed.

**Set Up the Workshop for WebLogic Platform Design-Time Environment**

After you create the `myworklist` domain, you can set up the design-time environment for the loan processing task plan by performing the following steps:

1. Click **File→New→Server**. The **New Server** dialog box appears (see Figure 2-5).
2. In the **New Server** dialog box, accept the default settings and click **Next**.

3. Browse and select the **myworklist** domain, which you created using the Configuration Wizard. It is located at `C:\bea\user_projects\domains\myworklist` (see Figure 2-6).
Create a New Worklist Application

A Worklist application consists of an EAR and a Web project, which contain all the files and directories that relate to a executable and self-contained Worklist application on the server.

The EAR project corresponds to the Enterprise Application. It hosts the Worklist system instance and the loan processing task plan for Acme Financial System. You will build and deploy this project to create the task scenario described in the overview of this tutorial.

The Web project, an instance of the Worklist user portal that acts as the user interface for the Worklist system (hosted by the EAR project). The Web project is a part of the EAR project.

To create a new Worklist application:

1. In BEA Workshop for WebLogic Platform, click File→New→Project.
   
   The New Project dialog box appears.

2. Select the WebLogic Integration→Worklist Application in the New Project dialog box (see Figure 2-7).
3. Click Next. The New Worklist Application dialog box appears.

4. Select the Process Host option from the Choose Worklist Application Type section, this will allow to use Worklist and Business Process Management together in the same application.

5. Specify the EAR Project Name as Loan_EAR in the EAR Project Name box.

6. Specify the Web Project Name as Loan_Web in the Web Project Name box.

7. Select the Create Utility Project check box and specify the Utility Project Name as LoanUtil. This project will contain all the WebLogic Integration schemas.

8. Select the Add WebLogic Integration System and Control schemas in utility project check box (see Figure 2-8).
9. Click **Finish** and the **Open Associated Perspective?** dialog box is displayed.

10. In the displayed **Open Associated Perspective?** dialog box, select the **Remember my decision** check box and click **Yes**. In doing so, you associate the project with the Task Plan perspective (see Figure 2-9).

**Figure 2-9 Open Associated Perspective? Confirmation Box**

You can see the Task Plan icon on the top right corner of the BEA Workshop for WebLogic Platform window.
Configure Users and Groups for Loan Processing

For Acme Financial System, the users and groups listed in Table 2-2 need to be created. To do this, you need to start the server created in the previous section if it is not started already. Perform the following to start the server afresh.

1. Select the server in the Servers pane (see Figure 2-10).

2. Right click and select the Start option.
Create Groups for the Loan Processing Task

Perform the following steps to create Group:

1. Open the Worklist Console using Run→Weblogic Integration→Worklist Console menu in the BEA Workshop for WebLogic Platform or alternatively open a Web browser, such as Internet Explorer, and enter the Worklist console URL:

   http://host:port/worklistconsole

2. Use the following credentials to log in to the WebLogic Integration Worklist console with administrator rights:

   - User name: weblogic
   - Password: weblogic

3. Click Worklist Users from the left panel.

4. Click Groups from the Worklist Users section in the left panel.

5. Click Add Group to open the Add New Group page (see Figure 2-11).

6. Specify the following details for the new group:

   - Group name: loanOfficer
   - Authentication Provider: SQLAuthenticator

---

Table 2-2  Users and Groups for Loan Processing

<table>
<thead>
<tr>
<th>Users</th>
<th>Groups</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>loanOfficer</td>
</tr>
<tr>
<td>Joe</td>
<td>loanOfficer</td>
</tr>
<tr>
<td>Mary</td>
<td>loanManager</td>
</tr>
<tr>
<td>Mark</td>
<td>loanManager</td>
</tr>
</tbody>
</table>
Figure 2-11  Add New Group Page

7. Click **Save**.

8. Click **Add Group** to create another group.

9. Specify the following details for the new group:
   - **Group name:** loanManager
   - **Authentication Provider:** SQLAuthenticator

10. Click **Save**.

After you create the groups, you need to create the users and assign the users to these groups.

**Create Users and Assign to Groups**

1. Click **Worklist Users** from the left panel.

2. Click **Add User**. This opens the Add New User - General Configuration page.

3. Specify the following details:
   - **Name:** John
   - **Provider:** SQLAuthenticator
   - **Password:** password
Note: The password must be of at least 8 characters.

4. Move **loanOfficer** from the list of Available Groups to the list of Current Groups (see Figure 2-12).

**Figure 2-12 Add New User - General Configuration**

5. Click **Save**.

6. This displays the Summary of Users page, which lists **John** as a user.

7. Repeat step 2 to step 5 for Joe.

8. For users Mary and Mark, select the **loanManager** group after repeating step 2 and step 3.
   
   **Note:** While you can change the password for these users, retain the same password for this tutorial.

9. Click **Save**.

**Figure 2-13** shows the summary of users and groups required for the loan processing task plan.
Figure 2-13  Summary of Users
Step 2: Modeling and Deploying the Loan Processing Task Plan

A *task plan* defines the business-specific life cycle to complete a task. A loan processing task plan depicts the multiple human interaction steps involved in processing a loan. For example, when a prospective customer submits a loan application, a loan officer needs to be assigned the task of checking the customer details and then approve the loan or forward it to the loan manager for further scrutiny. This step is a part of the task plan, which ensures that whenever a new loan application is submitted, a loan officer claims the task and processes the loan application.

In WebLogic Integration Worklist 9.2, a task plan can be modeled using the Workshop for WebLogic design-time environment and then be deployed to run on the server. Once the task plan is deployed, the task instances can be created by authorized systems or human entities. *Task Instances* or tasks are based on the task plan.

In this step, you will model and deploy the loan processing task plan using Workshop for WebLogic Platform design-time environment.

### Model and Deploy the Loan Processing Task Plan

The loan processing task plan will be modeled using Workshop for WebLogic. To model the task plan, you need to perform the following tasks:

- Create a New Task Plan
- Define the Steps for the Loan Processing Task Plan
- Define Actions in the Task Plan
- Define Constructors for the Task Plan
Create a New Task Plan

To create the loan processing task plan:

1. In the Package Explorer pane, right-click the Loan_EAR\EarContent folder, and select New→Folder.

2. In the New Folder dialog box, specify the folder name as Loan and click Finish to continue.


   The New Task Plan dialog box appears.

4. In the New Task Plan dialog box enter loan_approval in the File name (see Figure 3-1).

   Figure 3-1 New Task Plan

5. Click Finish to proceed.

Define the Steps for the Loan Processing Task Plan

A task plan is a collection of steps that define the action a human needs to perform when working through a task. For Acme Financial loan processing system, the steps involved are listed in Table 3-1.
Table 3-1  Steps in the Loan Processing Task Plan

<table>
<thead>
<tr>
<th>Step Name</th>
<th>Default Assignee</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Officer Review Pending</td>
<td>Loan Officer</td>
<td></td>
</tr>
<tr>
<td>Manager Review Pending</td>
<td>Loan Manager</td>
<td></td>
</tr>
<tr>
<td>Loan Approved</td>
<td></td>
<td>Step Completed</td>
</tr>
<tr>
<td>Loan Rejected</td>
<td></td>
<td>Step Aborted</td>
</tr>
</tbody>
</table>

To add these steps to the loan_approval task plan:

1. From the Palette box, click Step and then click anywhere in the *loan_approval.task tab to add a step. The default name for a new step is Step#, where # is an incremental numeric value that changes depending on the number of existing steps in the task plan.

2. Click the step again and change the name to OfficerReviewPending.

3. With the OfficerReviewPending step selected, click Assignment Instructions in the Properties tab.

4. In the Value column, click . This displays the Assignment Instructions dialog box, as shown in Figure 3-2.

5. Click Add and click the Name column to enter loanOfficer.

6. In the Type column, click the list box and select Group.

7. Select DEFAULT from the Candidate List Handling list box.

8. Click OK. The OfficerReviewPending step is now assigned to the loanOfficer group (see Figure 3-2).
9. Add another step to the loanApproval.task file and call it ManagerReviewPending.

10. With the ManagerReviewPending step selected, click Assignment Instructions in the Properties tab.

11. In the Value column, click . This displays the Assignment Instructions dialog box.

12. Click Add and click the Name column to enter loanManager.

13. In the Type column, click the list box and select Group.

14. Select DEFAULT from the Candidate List Handling list box.

15. Click OK. The OfficerReviewPending step is now assigned to the loanManager group.

16. From the Palette tab, click Complete Step and drop it in the loanApproval.task tab.

17. Change the name of the step to LoanApproved.

18. From the Palette tab, click Abort Step and drop it in the loanApproval.task tab.

19. Change the name of the step to LoanRejected.

After you add all the steps, the loanApproval.task tab will be displayed (Figure 3-3).
Define the User Properties of the Task Plan

User properties are business-specific data elements of a task plan. For the loan processing scenario, you need to define the user properties mentioned in Table 3-2.

Table 3-2 User Properties for Loan Processing Task Plan

<table>
<thead>
<tr>
<th>User Property Name</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LoanAmt</td>
<td>Integer</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td>SSN</td>
<td>String</td>
</tr>
<tr>
<td>Collateral Assets</td>
<td>String</td>
</tr>
<tr>
<td>Notes</td>
<td>String</td>
</tr>
</tbody>
</table>
**Note:** User properties are global and apply to all the steps throughout the life cycle of the task plan.

To create user properties:

1. From the User Properties tab, click the **Create user property** icon (see Figure 3-4).

**Figure 3-4 User Properties Tab**

2. In the Create User Property dialog box, enter the name of the user property as **LoanAmt**, and provide a brief description in the **Description** field.

3. Click the **Type** drop-down list and select **Integer** (see Figure 3-5).
4. Click **OK** to implement the new property.

5. Repeat step 1 to step 4 for the other properties listed in Table 3-2.

**Define Actions in the Task Plan**

Every step (other than terminal steps including Completed Step and Aborted Step) of the loan processing task plan can include actions, which allow the transition of the task instance from one step to another. The actions can be taken by authorized employees or system actors.

For the loan processing task plan, create the actions listed in Table 3-3.

**Table 3-3  Actions for the Loan Processing Task Plan**

<table>
<thead>
<tr>
<th>Step/Constructor</th>
<th>Action</th>
<th>User Properties Required</th>
<th>Resulting Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Officer Review Pending</td>
<td>Approve</td>
<td>Notes</td>
<td>Approved</td>
</tr>
<tr>
<td>Officer Review Pending</td>
<td>Reject</td>
<td>Notes</td>
<td>Rejected</td>
</tr>
</tbody>
</table>
To create the actions listed in Table 3-3:

1. Click **Action** in the Palette tab and drop it on the **OfficerReviewPending** step.
2. Change the action name to **Approve**.
3. In the Properties tab, click the **Required User Properties** $\rightarrow$ **Notes** property.
4. In the Value column, click $\text{Edit}$ to open the Property Notes dialog box.
5. Select the **Required** check box and click **OK**.
6. Click **Connections** in the Palette tab.
7. To create a connection between the **Approve** action and the **LoanApproved** step, which appears in green color, click the **Approve** action box.
8. Move the mouse over the **LoanApproved** step and click again. This creates the connection between the **Approve** action and the **LoanApproved** step.
9. Repeat step 1 to step 8 for the other steps listed in Table 3-3.

After you create the connections for each action, the Outline tab will look similar to Figure 3-6.
Define Constructors for the Task Plan

In a task plan, there is at least one constructor that defines how a task instance comes into existence. A constructor for a task plan lists the initial data to be provided for the creation of a task instance as well as the resulting step of the task instance. Each constructor needs to have a step associated with it. There may be more than one constructor for a task plan.

For the loan processing task plan, define the NewLoan constructor. This constructor will be used to create a loan task when a loan request comes in without a credit score and pre-approval.

**Note:** Constructors can be invoked by authorized employees or system actors, so that the loan task instances can be created either by human data entry or system execution.

To configure the constructor:

1. Click **Constructor** on the Palette tab and drop it in the Constructor container of the *loan_approval.task* tab.

2. Name the constructor **NewLoan**.
3. In the Properties tab, set the value for the LoanAmt, Name, and SSN properties to Required, by performing the following steps:

   a. In the Properties tab, click the LoanAmt property (see Figure 3-7).

   b. Click in the Value column to open the Property LoanAmt dialog box.

   c. Select the Required check box and click OK.

   d. Repeat step a to step c for the Name and SSN properties.

   For a new loan application, you need to create a connection between the NewLoan constructor and the OfficerReviewPending step.

4. Click Connection in the Palette tab to connect NewLoan to OfficerReviewPending.

5. Click the NewLoan constructor box and move the mouse over to the OfficerReviewPending step and click again.

   After you map the NewLoan constructor to OfficerReviewPending step, the Outline tab will appear similar to Figure 3-8.
6. Select File→Save All menu option to save the application before you proceed.

**Validate the Task Plan**

The final stage in designing and deploying the task plan is to validate if the task plan is working according to the required enterprise model specification.

To validate the loan processing task plan:

1. Click Worklist→Validate Task Plan for Runtime.

2. If the task plan is valid, then the Validation Results dialog box appears (see Figure 3-9).

**Figure 3-9 Validation Results Dialog Box**

3. Click OK to confirm.
4. Select **File→Save All** menu option to save the application before you proceed.

**Deploy the Loan Processing Task Plan**

Once the loan processing task plan is modeled completely, you can deploy it on WebLogic Integration Server.

To deploy the loan processing task plan:

1. In the Package Explorer pane, right-click the **Loan_Web** project that you created previously and select **Run As→Run on Server**. The Run on Server dialog box is displayed (see Figure 3-10).

![Run on Server Dialog Box](image)

**Figure 3-10 Run on Server Dialog Box**

2. Select **Choose an existing server** and from the Select the server that you want to use: list, select the **myworklist** server and click **Next**.

   This will display the Add and Remove Projects dialog box.

   **Description:** BEA WebLogic v3.2 Runtime

   Set server as project default (do not ask again)

   ![Add and Remove Projects Dialog Box](image)
3. Ensure that Loan_EAR is listed in the Configured projects list. If it is not then select Loan_EAR from the Available projects list, and click Add.

4. Click Finish to start deploying the project on the server.

It will take some time to deploy the project on the server. After the task plan is deployed successfully, it opens up on the Worklist User Portal within the BEA Workshop for WebLogic Platform (see Figure 3-11).

Figure 3-11 Worklist User Portal
Step 3: Testing the Task Plan Using Worklist User Portal

The user portal provides Worklist users an interface for accessing the task instances that they are authorized to deal with. In the user portal, the user will see list of task instances associated with them or the groups they belong to. These lists should be considered to the user’s Inbox for Worklist.

There Upcoming Tasks and Overdue Tasks portlets show tasks created by the user or owned by the user. Task claimed by the user are shown with a special icon containing a check mark. This is done to clearly indicate the tasks this user is expected to work on (by taking actions on them and setting their properties). Tasks that are were created by or are owned by the user require the user to keep track of them, but not necessarily to work on them.

The Assigned Tasks portlet shows tasks that are assigned to the user, or the groups to which the user belongs, but are not claimed by anyone. The claimed tasks are claimed by the user and to be worked on by the user only. For example, if loan officer John claims a loan processing task instance, then loan officer Joe will not have access to this task instance, and will not see the task in his view of the user portal.

In this step, you will perform the following tasks:

- Create the Loan Processing Task Instance
- Claim the Loan Processing Task Instance
- Reject the Loan Task Instance
Create the Loan Processing Task Instance

To test the newly deployed loan processing task plan, you need to create a new task instance. Before integrating with the system application, the task plan can be tested by creating a loan processing task instance.

The `NewLoan` constructor will be used to create the loan processing task instance. To create the task instance:

1. Open a Web browser and enter the following URL to open the Worklist User Portal test browser:
   
   `http://localhost:7001/Loan_Web`

   **Note:** You can use any external browser, for example Internet Explorer, or the default browser that comes with BEA Workshop for WebLogic Platform.

2. Log in to the `Loan_Web` project using the following credentials:
   
   **Username:** weblogic
   **Password:** weblogic

   The Home page is displayed with the Inbox’ for the user. This is a portal page with portlets for the Inbox of overdue, upcoming, and assigned tasks, along with the portlet that allows you to create a new task (see Figure 4-1).
Create the Loan Processing Task Instance

3. Click the /Loan/loan_approval 1.0 option in the Create Task portlet. The Create New Task page is displayed.

4. Ensure that NewLoan is selected as the task plan constructor.

5. Specify the details listed in Table 4-1 for the other fields on the page:

Table 4-1 Specifications for the New Loan Approval Task Instance

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Name</td>
<td>Car loan for Maggie May</td>
</tr>
<tr>
<td>User Properties: SSN</td>
<td>222-33-4444</td>
</tr>
<tr>
<td>User Properties: LoanAmt</td>
<td>10000</td>
</tr>
<tr>
<td>User Properties: Name</td>
<td>Maggie May</td>
</tr>
</tbody>
</table>

6. Click Create Task. The task is created and shows up in the Upcoming Tasks portlet on the home page, as shown in Figure 4-2.
Figure 4-2  Upcoming Tasks Portlet After Creating a Task

![Upcoming Tasks Portlet](image)

Note: The task shows up on the Inbox of the user weblogic because weblogic is the owner of the task instance. By default, the user who creates the task instance becomes the owner of the task instance. This enables the owner to track the status of the task instance although the owner is not assigned to work on the task instance.

7. Click Logout to close and log out as weblogic from the Worklist User Portal.

**Claim the Loan Processing Task Instance**

The new task instance shows up in the list of upcoming tasks, which implies that a loan officer needs to claim the task and process it. Loan officer John will claim this task and work on it.

To claim the task instance, *Car loan for Maggie May*:

1. Start a new session of the loan web project using the URL:
   
   http://localhost:7001/Loan_Web

2. Log in to the portal using the following credentials:

   **Username**: John
   
   **Password**: password

   Because the task instance, *Car loan for Maggie May*, is assigned to the loanOfficers group, it will show up on the Assigned Tasks portlet of John’s Inbox (see Figure 4-3).
Figure 4-3  Task Assigned to User

3. Click **Claim** to claim the particular task. This will move the task from the Assigned Tasks portlet to the Upcoming Tasks portlet for user John, as shown in Figure 4-4.

**Figure 4-4  Upcoming Tasks for Loan Officer John**

As the task instance has been claimed by John, it will no longer show up in Joe’s Assigned Tasks portlet. Joe is the other loan officer who could have claimed the task.

4. Click **Car Loan for Maggie May** in the Upcoming Tasks portlet. This will display the Task Work page with the task details, and the Action options available for user John. As show in Figure 4-5.
5. Select Request Manager Review in the Actions section to forward the request to the loan managers group for approval, and click Next.

6. In the Key Action Properties of the refreshed Web page that appears, enter the string Loan amount of 10,000 sent for sanction by loan managers.

7. Click Submit. The task is now assigned to the loan managers group and will not show up in John’s Inbox.

8. Logout as user John from the Worklist User Portal.

**Reject the Loan Task Instance**

After John forwards the new loan application for approval to the managers, one of the managers needs to claim the task, decide to approve or reject the loan, and the system will process the request accordingly. Perform the following steps to claim the task instance and subsequently to reject the loan request.
1. Start a new session of the Loan_Web project at the following URL:
   http://localhost:7001/Loan_Web

2. Log in to the portal using the following credentials:
   
   **Username:** Mary
   **Password:** password

3. As the task instance, *Car loan for Maggie May*, has been passed on to the loanManagers group, it will show up on the Assigned Tasks portlet of Mary’s Inbox. The Assigned Tasks portlet will look similar to **Figure 4-3**.

4. Click **Claim** to claim the particular task. This will move the task from the Assigned Tasks portlet to the Upcoming Tasks portlet for user Mary (**Figure 4-6**).

**Figure 4-6 Task Instance in Upcoming Tasks Portlet of Mary’s Inbox**

![Figure 4-6 Task Instance in Upcoming Tasks Portlet of Mary’s Inbox](image)

As the task instance has been claimed by Mary, it will no longer show up in Mark’s Inbox. Mark is the other loan manager who could have claimed the task.

5. Click **Car Loan for Maggie May** in the Upcoming Tasks portlet. This will display the Task Work page with the task details, and the Action options available for user Mary. (see **Figure 4-7**).
6. In the Actions section, select **Reject** to reject the loan as shown in Figure 4-7 and click **Next**.

7. In the Key Action Properties of the refreshed Web page that appears, enter the string **Loan rejected on bad credit**.

8. Click **Submit** to complete the task. As the loan has been rejected, the task instance will no longer appear in Mary’s Inbox.

9. Logout as user Mary from the Worklist User Portal.
Step 4: Managing Task Instances Using Worklist Console

This section describes how to use the Worklist Console to modify a task in the Loan_Web project. The key objectives for this section are:

- Log in with administrator rights
- Re-assign a pending task
- Validate the update using the Worklist User Portal

This section is an extension of the previous section, and relies on its environment. So, it is assumed that you have the Loan Application open in Workshop for WebLogic Platform, and the myworklist server is up and running.

Update the Application Using Worklist Console

Perform the following steps to log in and re-assign a task:

1. Open a Web browser and enter the following URL to open the WebLogic Integration Management Console:

   http://localhost:7001/worklistconsole

   Alternatively, select Run→WebLogic Integration→Worklist Console to open the WebLogic Integration Management Console in the Workshop for WebLogic Platform IDE.

2. Use the following credentials to log in to the Worklist Console, with administrator rights:

   Username: weblogic
   Password: weblogic
3. Click View Tasks for the Loan_EAR Worklist System Instance as shown in Figure 5-1. Worklist Application Management: Worklist System Instance Page

As a user with administrator rights, you can view all the Worklist Instance details at any given point. After clicking View Tasks, the page is refreshed and all the tasks in the Loan_EAR project are listed in the Task Summary page.

4. In the Task Summary page, select the task from the list by clicking the check box adjacent to the task name. This task has been claimed by Mary, as shown in Figure 5-2.

5. Select Reactivate from the dropdown menu in Apply this action on selected tasks and click Submit.

6. Click OK to proceed.

Selected action REACTIVATE executed successfully dialog appears in the Tasks Summary - Loan_EAR page.

Note: Since Mary, rejected the loan in the previous chapter, you have to activate the task.

7. Click Assign to update the task with the intent to re-assign it to user weblogic.
Note: At this point, if you log in to the Worklist User Portal as *weblogic*, you will notice that there are no tasks assigned to you. After completion of this section, the list of tasks assigned should be updated to reflect re-assignment of the task selected in Figure 5-2.

8. In the refreshed page, move *weblogic* from the *Users* list to the *Selected Users* list, and *Administrators* from the *Group* list to the *Selected Groups* list (see Figure 5-3).

**Figure 5-2 Assign Task to User weblogic**

9. Click **Assign** to complete this task and the refreshed page displays the Task Summary page. As highlighted in Figure 5-4, the task has been updated as the **Claimant** field is empty. This task was claimed by Mary prior to this re-assignment exercise.
10. Click **Logout** to close the Worklist Console and proceed to the next section, validating the re-assignment.

**Verify Updated Application in Worklist Portal**

After completing the re-assignment task, verify if the task has been assigned to the user **weblogic** using Worklist Portal.

1. Start a new session of the loan web project in a Web browser using the URL:  

2. Log in to the portal using the following credentials:  
   **Username:** weblogic  
   **Password:** weblogic

3. Click **Login** to display the Task home page for user **weblogic**, as shown in Figure 5-6.
Figure 5-4  View Task Home Page for User weblogic

The task instance *Car loan for Maggie May* is now displayed in your Assigned Tasks portlet.
Step 5: Using JPDs with Worklist

This section describes details on how to use JPDs and Worklist controls to support the integration of business processes with human actors via the Worklist system.

As with other built-in controls in Workshop for WebLogic Platform, you use the controls by adding instances of the controls to your business process. Subsequently, you invoke operations on the controls at the point in the business process at which you want to reach out to one or more human actors.

In this step, you will perform the following tasks:

- Subscribe to Worklist Events
- Configure a Perform Node
- Verify the Worklist Event is Published
- Use the Worklist Control

Information in this section is an extension of the previous section. So, it is assumed that the Loan Application is open, and the myworklist server is up and running.

Subscribe to Worklist Events

Perform the following steps to configure access to the Message Broker Channel, and initiate the access using a start event.

1. In the Package Explorer pane, right-click the Loan_Web\src folder, and select New→Folder. This will display the New Folder dialog box.
2. Enter **processes** in the **Folder name** field and click **Finish**.

3. Select the **processes** folder and use **Ctrl+N** to display the Select a Wizard dialog box.

4. Select **WebLogic Integration → Process** and click **Next** to display the New Process File dialog box.

5. In the **Name** field, enter **Loan_jpd**. This will create a JPD process file **Loan_jpd.java** under the newly created processes folder (see Figure 6-1).

Figure 6-1  Defining a New JPD File

![](image)

6. Click **Finish** to complete the process.

   The new JPD appears in the Design view (see Figure 6-2).
Figure 6-2  Creating a New JPD

7. Double-click the Select Start Event (see Figure 6-2).
   The node builder displays.

8. In the node builder, select **Subscribe to a Message Broker channel and start via an event...** and click **Close**. The JPD design view is refreshed and the Start node is named **Subscription**.

9. Click on **Subscription** and replace it with **Subscribe worklist event**.

10. Double-click the Start node to configure it (see Figure 6-3).
11. In the **General Settings** tab, select */WorklistEvent* as the **Channel Name**.

12. In the **Specify Filter** tab, select **eventType** from the **TaskEventMetadata** xml tree, and type **CREATE** as the filter value.

13. In the **Receive Data** tab, do the following:
   - select **Create new variable** next to **RawData x0** and in the Create Variable dialog box that appears, enter **worklistEventDa** in the **Variable Name** field, and select **com.bea.data.RawData** as its type.
   - select **Create new variable** next to **TaskEventMetadataDocumentx1** and in the Create Variable dialog box that appears, enter **worklistEventMetaData** in the **Variable Name** field, and enter **com.bea.wli.worklist.xml.TaskEventMetadataDocument** as its type.

14. Click **OK** to set the variable name, and click **Close** on the Start Node configuration to complete this step.

**Configure a Perform Node**

After creating a JPD to subscribe to a Worklist event to start, create and configure a Perform node to echo the event. Perform the following steps:

1. In the Design view, select **Insert→Perform** to insert an action node between Subscription and the Finish nodes.
2. Name the node as **Print Messages**, as shown in Figure 6-4.

![Figure 6-4 Adding the Perform Node](image)

3. Click the **Source** tab of the JPD and it should highlight the perform method definition.

4. Enter the following code into the perform method:
   ```java
   System.out.println("####Got worklist event for loan task type");
   ByteArrayInputStream bais = new ByteArrayInputStream(worklistEventData.byteValue());
   ObjectInputStream ois = new ObjectInputStream(bais);
   TaskEvent event = (TaskEvent)ois.readObject();
   System.out.println("####Got TaskEvent data in JPD: "+event.getSummary());
   ```

5. Enter the following import statements to the beginning of the JPD.
   ```java
   import java.io.ByteArrayInputStream;
   import java.io.ObjectInputStream;
   ```
import com.bea.wli.worklist.api.events.data.*;

6. Select File→Save or use Ctrl+S to save the file.

**Deploy the Loan_JPD**

1. In the Package Explorer pane, right-click the Loan_jpd.java project that you created previously and select Run As→Run on Server.
   
   The Run on Server dialog box is displayed.

2. Select Choose an existing server and from the Select the server that you want to use: list, select the myworklist server and click Next.
   
   This will display the Add and Remove Projects dialog box.

3. Click Next.

4. Ensure that Loan_EAR is listed in the Configured projects list. If it is not then select Loan_EAR from the Available projects list, and click Add.

5. Click Finish to start deploying the project on the server.

**Verify the Worklist Event is Published**

Test the Application by creating a new task instance using the Worklist User Portal. After the task instance has been created, the Worklist event is published to the Message Broker, and a JPD instance is started by the event. Perform the following steps:

1. Start a new session of the loan web project in a Web browser using the URL:
   
   http://localhost:7001/Loan_Web

2. Log in to the portal using the following credentials:
   
   **Username:** weblogic
   
   **Password:** weblogic

3. Click **Login** to display the Task home page for user weblogic (see Figure 6-5).
4. Click the /Loan/loan_approval 1.0 option in the Create Task portlet. The Create New Task page is displayed.

5. Ensure that NewLoan is selected as the task plan constructor.

6. Specify the details listed in Table 6-1 for the other fields on the page:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Name</td>
<td>Car loan for Maggie JPD</td>
</tr>
<tr>
<td>User Properties: SSN</td>
<td>222-33-4444</td>
</tr>
<tr>
<td>User Properties: LoanAmt</td>
<td>40000</td>
</tr>
<tr>
<td>User Properties: Name</td>
<td>Maggie JPD</td>
</tr>
</tbody>
</table>

7. Click Create Task. The task is created and shows up in the Upcoming Tasks portlet on the home page.

You can also verify creation of the JPD instance by logging into the WebLogic Worklist Console.

1. Open a Web browser and enter the following URL to open the WebLogic Worklist Console:
   http://localhost:7001/worklistconsole
2. Use the following credentials to log in to the Worklist Console, with administrator rights:

   **Username:** weblogic
   **Password:** weblogic

3. Click **View Tasks** for the Loan_EAR Worklist System Instance.

4. In the Task Summary page, select the task from the list by clicking the check box adjacent to the task name, Car Loan for Maggie JPD.

   The Worklist Task Details is displayed (see **Figure 6-6**).
In this section, you will create a loan processing task instance using a Task Control in a JPD. During this process, you will create a Worklist JPD (WorklistControl), a Task Control (MyControl), deploy the process, and subsequently create a sample task to validate the task instance creation.
Create a Worklist JPD

This section details steps on how to create a Worklist JPD.

1. In the Package Explorer pane, select the Loan_Web\src\processes folder, and use Ctrl+N to display the Select a Wizard dialog box.

2. Select WebLogic Integration→Process and click Next to display the New Process File dialog box, similar to Figure 6-1.

3. In the Name field, enter WorklistControl. This will create a JPD process file WorklistControl.java under the processes folder.
   
   Click Finish to complete the process.

   The new JPD appears in the design view (see Figure 6-7).

Figure 6-7 New JPD Using Worklist Control

4. Double-click Select Start Event (see Figure 6-7).
   
   The node builder displays.

5. In the node builder, select Invoked via a Client Request and click Close. The JPD design view is refreshed and the Start node is named Client Request.
6. Double-click **Client Request** node to invoke the node builder for the Client Request node as shown in **Figure 6-8**.

**Figure 6-8 Configuring Client Request**

7. In the **General Settings** tab, click **Add** to display a dialog box for defining parameters. Create three parameters and configure their types as shown in **Table 6-2**.

**Table 6-2 Setting Parameters for the Client Request**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskName</td>
<td>String</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
</tr>
<tr>
<td>SSN</td>
<td>String</td>
</tr>
<tr>
<td>LoanAmt</td>
<td>java.lang.Long</td>
</tr>
</tbody>
</table>

8. Click the **Receive Data** tab to create new variables and assign them the respective parameters created in the previous step. The variable assignment details are shown in **Figure 6-9**.
The four new variables of default type are: `TaskName(String), Name(String), SSN(String), and LoanAmt(Long)`.

9. Click **Close** to continue, and the JPD appears as shown in Figure 6-10.

**Figure 6-10  WorkListControl JPD with Client Request Start Node**

### Create a Task Control

In this section, you will create a Task Control that will trigger the creation of a task instance.

1. In the Package Explorer pane, select the `Loan_Web\src\processes` folder, and use **Ctrl+N** to display the Select a Wizard dialog box.

---

**Figure 6-9  Assigning Variables to Parameters**

![Variable Assignment](image-url)
2. Select **WebLogic Integration Controls → Task** and click **Next** to display the Create Control page of the Insert Control: Task dialog box.

3. Enter **MyControl** in the Name field and ensure the other fields have same value as shown in Figure 6-11.

![Figure 6-11 Creating a Task Control](image)

4. Click **Next** to proceed to the Task Plan page of the Insert Control: Task dialog box.

5. For the Task Plan field, click **Browse** and select `Loan_EAR\EarConten\Loan\loan_approval.task` as shown in Figure 6-12.
6. Click OK to continue.

7. Click Finish to add the new Task Control to the JPD.

8. Select MyControl.java from the Package Explorer, drag and drop it to Controls folder in the Data Palette.

Note: If the Data Palette pane is not visible, go to Window→Show ViewOther→WebLogic Integration→Data Palette

9. Select File→Save or use Ctrl+S to save the JPD.

Add Task Plan Constructor to JPD

In this section, you will add the task instance creation constructor to the Worklist JPD.

1. Right click the WorklistControl.java file in the Package Explorer pane and select Open With→Process Editor option. Ensure the JPD is displayed in the Design tab and that you are using the Process Perspective.

2. From the Data Palette pane on the bottom right corner of the IDE, navigate to Controls→myControl and select the OfficerReviewPending NewLoan(String taskName, String SSN, Long LoanAmt, String Name) method. As shown in Figure 6-13.
3. Drag and drop the selected method into the **WorklistControl** JPD, between the **Client Request** and the **Finish** nodes. The method will be added as **NewLoan** (see **Figure 6-14**).
4. Double click **NewLoan** and configure the **Send Data** properties, as shown in **Figure 6-15**.
   You can leave the General Settings and Receive Data properties as they are.

**Figure 6-15  Mapping the Send Data Variables with the Control Parameters**

5. Click **Close** to implement the settings.

6. Select **File→Save** or use Ctrl+S to save the JPD.
Validate the WorklistControl JPD

In this section, you will deploy the JPD and after successfully deploying the JPD, you will validate it using test values.

1. Ensure the WorklistControl.java is selected in the Package Explorer, and click the Run→Run As option. After successful deployment the JPD process page will be launched in the IDE browser.

2. Click the Test Form tab of the WorklistControl process browser, as shown in Figure 6-16.

Figure 6-16 WorklistControl Process Test Form Page

3. Enter the test values in their respective fields, as shown in Table 6-3.

Table 6-3 Test Values for the WorklistControl Process

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskName</td>
<td>Loan for John Doe</td>
</tr>
<tr>
<td>Name</td>
<td>John Doe</td>
</tr>
</tbody>
</table>
4. Click ClientRequest to execute the process with the test values. After a successful execution, the TestForm tab is refreshed (see Figure 6-17).

Figure 6-17 Successful Execution of the WorklistControl JPD

5. Start a new session of the Loan_Web project in a Web browser using the URL:
   http://localhost:7001/Loan_Web

6. Log in to the portal using the following credentials:
   - **Username**: John
   - **Password**: password

   The user John is part of the loanOfficer group and the task instance created should be visible in his Assigned Tasks portlet box.

### Table 6-3 Test Values for the WorklistControl Process

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>123-456</td>
</tr>
<tr>
<td>LoanAmt</td>
<td>20000</td>
</tr>
</tbody>
</table>

6-18 Tutorial: Building a Worklist Application
7. Click **Login** to display the Task home page for user John, as shown in **Figure 6-18**.

**Figure 6-18  Task Home Page for User John**

The **Assigned Tasks** portlet box displays the **Loan for John Doe** task. This confirms that a task instance was created by a JPD using a Control.
Advanced Topic: Adding a Customized User Interface

This chapter describes how to create a customized task user interface (for use in the Worklist user portal). Worklist provides a default task user interface (shown in the Work on Task page of the user portal). This user interface dynamically creates forms based on the task plan metadata. For example, the default task user interface consults the current step for a task before deciding what actions to make available on the ‘take action’ page, and consults the properties defined for an action before deciding what properties to show on the ‘complete task action’ page. This allows you to perform most human interaction in Worklist without any custom user interface development.

However, there may be instances where you need to customize the interface and control what is displayed for a given step, or for the entire task plan. Worklist enables you to provide a customized user interface for tasks based on a given task plan (and optionally a specific step within a task plan). This allows you to integrate custom business logic, external systems, etc. into the processing of task actions and property settings.

Your custom task user interface is used in place of the default Worklist-supplied task user interface when viewing tasks based on task plans (or steps of those plans) you designate. It will appear in place of the default task user interface on the ‘Work on Task’ page of the Worklist user portal. This granular replacement of the default task user interface allows you to specify a custom task user interface only where it is needed, and use the default task user interface everywhere else.

For example, a Loan Manager may need to check the credit rating of the customer before approving or rejecting the loan. Using the custom task UI, you can customize the user portal to display information that will empower the Loan Manager to make a well informed decision.

The following topics are covered in this chapter:
Define Web Page Mock-Up and Flow

Before you start creating a customized user interface, define the appearance of the page by creating a mock-up. For this tutorial, create mock-up pages for the “Manager Review Page” and the “Asset Summary Page” (See Figure 7-1 and Figure 7-2).

Figure 7-1  Manager Review Mock-Up Page

| Customer Name: John W. Smith | This comes from task properties |
| SSN: 111-11-1111 | This comes from task properties |
| Loan Amount: $10,000 | This comes from task properties |

View Asset Summary  Forward to Asset Summary Page

Reason for Action: Bad credit  From/To task properties
Collateral Assets:  From/To task properties

Approve  Reject  Actions on the Step
Now that the mock-up is complete, proceed with defining the page flow as described in the following section.

**Create the Page Flow**

After determining the appearance of the customized user interface pages, define the logic and use of these pages as follows:

1. In the Package Explorer pane, right-click the Loan_Web\src folder, and select New→Other. This will display the Select a Wizard dialog box.

2. Select Web→Page Flow and click Next (see Figure 7-3).
3. The **New Page Flow** dialog box appears, enter **manager** in the Page Flow Folder name field and **ManagerReview** as the Controller name.

4. Select the **Make this a nested page flow** check box and click **Finish** (see Figure 7-4).
The **Open Associated Perspective?** dialog box is displayed.

5. In the displayed **Open Associated Perspective?** dialog box, select the Remember my decision check box and click **Yes**. In doing so, you associate the project with the Page Flow perspective.

6. The Page Flow Editor view appears (see **Figure 7-5**).
Edit the page flow as follows:


2. Delete 
   ```java
   simpleActions = { @Jpf.SimpleAction(name = "begin", path = "index.jsp") }
   ```

   The initial view in the page flow was as follows:
   ```java
   @Jpf.Controller(nested = true, simpleActions = { @Jpf.SimpleAction(name = "begin", path = "index.jsp") })
   ```

   After editing it, it should be:
   ```java
   @Jpf.Controller(nested = true)
   public class ManagerReview extends com.bea.wli.worklist.TaskUIPageFlowController {
   ```

   **Note:** This change will result in some compilation errors saying ‘Action "begin" was not found.’ This error will be resolved in subsequent steps. You can view the compilation error in the ‘Problems’ view in the bottom part of the IDE. Make sure ‘Problems’ view is opened. To open the Problems view from the menu, go to **Window→Show View→Problems**.
Define Form Beans

You must define form beans to support the two web pages mocked up (see “Define Web Page Mock-Up and Flow” on page 7-2).

We create three form beans as inner classes of the Manager Review Pane, they are as follows:

- ManagerReviewForm - This will support the Manager Review web page.
- AssetSummaryForm - This will define the individual assets.
- AssetForm - This states the assets from the AssetSummaryForm.

Create Form Beans

1. In the Page Flow Explorer, right-click Form Beans and select New Inner Class Form Bean (see Figure 7-6).

![Figure 7-6 New Inner Class Form Bean](image)

2. A new form bean with the default name NewFormBean is created.
3. Right-click **NewFormBean → Rename**, and name it as **ManagerReviewForm**.

4. Repeat step 1 and step 3 and name it as **AssetSummaryForm** and **AssetForm** (see Figure 7-7).

**Figure 7-7  Form Beans**

Define **ManagerReviewForm**

1. The **ManagerReviewForm** includes the following variables and data type:
   - Name (String)
   - SSN (String)
   - LoanAmount (Int)
   - NotesProp (PropertyInstanceHolder)
   - CollateralAssets (String)

2. Select the above variables and place them in the **ManagerReview.java** source view.

3. Select the variables and select Source → **Generate Getters and Setters**.
   The **Generate Getters and Setters** dialog box appears.

4. Select the properties variable variables (Figure 7-8).
5. Click **OK**.

After you defined the variables, the class should look as the follows:

```java
@Jpf.FormBean
public static class ManagerReviewForm implements java.io.Serializable {
    private static final long serialVersionUID = 746621147L;

    private String _name;
    private String _ssn;
    private int _loanAmount;
    private PropertyInstanceHolder _notesProp;
    private String _collateralAssets;
```
public int getLoanAmount() { return _loanAmount; }
public void setLoanAmount(int loanAmount) { _loanAmount = loanAmount; }
public String getName() { return _name; }
public void setName(String name) { _name = name; }
public String getSsn() { return _ssn; }
public void setSsn(String ssn) { _ssn = ssn; }
public PropertyInstanceHolder getNotesProp() { return _notesProp; }
public void setNotesProp(PropertyInstanceHolder notesProp) {
    _notesProp = notesProp; }
public java.util.SortedSet<AssetForm> getAssets() { return _assets; }
public int getCreditScore() { return _creditScore; }

**Note:** The serialVersionUID value will differ. It is auto-generated and can be different from the one shown here.

**Define AssetSummaryForm**

1. The **AssetSummaryForm** includes the following properties and data type:
   - Name (String)
   - Ssn (String)
   - Assets (SortedSet<AssetForm>)
   - CreditScore (int)

2. Repeat **step 2** and **step 3** of ManagerReviewForm.

3. Select the property variables listed above.

4. Click **Ok**.

5. Enter the following code into the AssetSummaryForm:
   
   ```java
   public java.util.SortedSet<AssetForm> getAssets() { return _assets; }
   public int getCreditScore() { return _creditScore; }
   ```
After you defined the variables, the class should look as the following:

```java
@Jpf.FormBean
public static class AssetSummaryForm
    implements java.io.Serializable {
    private static final long serialVersionUID = 1517513921L;
    private java.util.SortedSet<AssetForm> _assets;
    private int _creditScore;
    private String _name;
    private String _ssn;
    public AssetSummaryForm() {
        _assets = new java.util.TreeSet<AssetForm>();
    }
    public String getName() { return _name; }
    public void setName(String name) { _name = name; }
    public String getSsn() { return _ssn; }
    public void setSsn(String ssn) { _ssn = ssn; }
    public java.util.SortedSet<AssetForm> getAssets() { return _assets; }
    public int getCreditScore() { return _creditScore; }
}
```

**Note:** The serialVersionUID value will differ. It is auto-generated and can be different from the one shown here.

In the AssetSummaryForm, add the following code to allow the form bean to load asset and credit score information.

This information is loaded in a very simplistic way (properties files) that is sufficient for the purposes of this tutorial. In a real application, this information would likely come by way of a Java API or web service to an external system.

The page flow action implementations use the loadSummaryInfo method included in the following code to initialize the AssetSummaryForm object with asset and credit score information for the user given by the name variable.

```java
public void loadSummaryInfo(HttpSession session) {
```

Tutorial: Building a Worklist Application
loadCreditScore(session);
loadAssets(session);
}
public int getTotalActualAssetValue() {
    int total = 0;
    for (AssetForm asset: _assets) {
        total = asset.getActualValue();
    }
    return total;
}
protected void loadCreditScore(HttpSession session) {
    // Load the credit scores as properties
    String resourceName = "/creditRatings/creditRatings.properties";
    java.util.Properties props = loadProperties(resourceName, session);
    _creditScore = getIntProperty(props, _name);
}
protected void loadAssets(HttpSession session) {
    // Load the assets as properties
    String resourceName = "/assets/" + _name + ".properties";
    java.util.Properties props = loadProperties(resourceName, session);
    String assetList = props.getProperty("assetList");
    if (assetList != null) {
        java.util.StringTokenizer st = new java.util.StringTokenizer(assetList, ",");
        while (st.hasMoreTokens()) {
            String assetName = st.nextToken().trim();
            AssetForm asset = new AssetForm();
            asset.setName(assetName);
int value =
    getIntProperty(props, assetName + "." + "value");
asset.setValue(value);
int amountOwed =
    getIntProperty(props, assetName + "." + "amountOwed");
asset.setAmountOwed(amountOwed);
_assets.add(asset);
}
}
protected java.util.Properties
loadProperties(String resourceName, HttpSession session) {
    // Load the resources as properties
    java.io.InputStream is = null;
    try {
        is = session.getServletContext().
            getResourceAsStream(resourceName);
java.util.Properties props = new java.util.Properties();
if (is != null) {
    props.load(is);
}
return props;
} catch (Exception e) {
    // TODO: Better handling
    e.printStackTrace();
} finally {
    if (is != null) {
        try { is.close(); } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
return new java.util.Properties();
}

private int getIntProperty(java.util.Properties props, String key) {
    String value = props.getProperty(key);
    if (value == null) {
        return 0;
    }
    return Integer.valueOf(value);
}

After completing the above steps, there will be a compilation error regarding the missing begin() method in the page flow.

**Define AssetForm**

1. The **AssetForm** will include the following properties and data type:
   - Name (String)
   - Value (int)
   - AmountOwed (int)
   - TotalValue (int)

2. Repeat step 2 and step 3 of ManagerReviewForm.

3. Select the property variables listed above.

4. Click Ok.

After you defined the variables, the class should look as the following:

```java
@Jpf.FormBean
public static class AssetForm
    implements java.io.Serializable, Comparable {
    private static final long serialVersionUID = 1491696939L;
    private String _name;
    private int _value;
```
private int _amountOwed;

public int getAmountOwed() {
    return _amountOwed;
}
public void setAmountOwed(int lienValue) {
    _amountOwed = lienValue;
}
public String getName() {
    return _name;
}
public void setName(String name) {
    _name = name;
}
public int getValue() {
    return _value;
}
public void setValue(int value) {
    _value = value;
}
public int getActualValue() {
    return _value - _amountOwed;
}

Note: The serialVersionUID value will differ. It is auto-generated and can be different from the one shown here.

After completing the above steps, there will be a compilation error stating that AssetForm class must implement the inherited abstract method Comparable.compareTo(Object). To resolve this error, copy the following method and paste it inside the AssetForm class. The code allows the asset summary web page to collate the individual asset items in descending total asset value.

    public int compareTo(Object o) {
if (!(o instanceof AssetForm)) {
    return 0;
}
AssetForm other = (AssetForm)o;
int otherActualValue = other._value - other._amountOwed;
return otherActualValue - getActualValue();

Add the following method to the end of the AssetForm class. This method will be used to retrieve an ‘actual’ asset value from the JSP pages we define later.

    public int getActualValue() {
        return _value - _amountOwed;
    }

**Define Support Asset Information Files**

Define information to support a loan request for a person named John Smith (use spelling and case as given here) as follows:

1. In the Package Explorer pane, expand Loan_Web.
2. Right-click WebContent→New→Folder.
   The New Folder dialog box appears (see Figure 7-9).
3. Enter **assets** in the Folder Name and click **Finish**.

4. Create another new folder, enter **creditRatings** in the Folder Name and click **Finish**.

5. Right-click **asset** and select a new file.
   
   The **New File** dialog box appears.

6. In the File Name enter **John Smith.properties** and click **Finish**.
   
   It appears in the editor.

7. Enter the following details in the editor:
   
   ```java
   assetList=Home, Car
   Home.value=300000
   Home.amountOwed=290000
   Car.value=20000
   Car.amountOwed=19000
   ```
8. Right-click **creditRatings** and select a new file.

9. In the File Name enter **CreditRating.properties** and click Finish.

10. Enter the following details (exactly as shown) on a single line in the properties file:

    John\ Smith=100.

**Define Actions on the Page Flow**

Define actions on the page flow to move between the Manager Review and Asset Summary pages, and to take the Approve and Reject actions on the task. Page flow actions are methods on the page flow controller that allow the UI to forward to new pages, optionally calculating results and passing form beans to the pages to which you forward.

Form beans are passed from an action to a web page in order to populate display fields on the page. Then, values from fields on the web page are collected and placed into properties on the form bean when the web page is submitted back to the server for processing.

Action methods can accept a form bean populated as the result of clicking a submit button on a web page, by defining the form bean as a parameter to the action method. For an example of this, see the ‘approve’ action below. Action methods can also pass a form bean on to a target web page to which the action is forwarding. This is done by passing a Forward object that has a form bean object set on it. For an example of this, see the ‘show asset summary’ action below.

Define the following actions for initialization:

- **begin**: Initialize this controller, and call the super class helper to initialize stuff we get for free. This includes task context, standard form beans for a task and action, and property editing support.

Define the following actions for page navigation:

- **viewAssetSummaryAction**: Navigates from the manager review page to the asset summary page. This action will take a ManagerReviewForm form bean, and will pass an AssetSummaryForm form bean on to the asset summary page (via a Forward object returned from the action method)

- **returnToManagerReviewAction**: Navigates from the asset summary form back to the manager review form. This action will take an AssetSummaryForm form bean, and will pass a ManagerReviewForm form bean on to the manager review page (via a Forward object)

**Note**: The above two actions are natural reciprocals of each other. This reflects their purpose to navigate between two pages in a cyclic fashion.
Define the following actions to handle user actions on the task:

- approveLoanAction: Calls the helper functions on the TaskUIPageFlowController controller class to take the ‘Approve’ action on the task. This action then exits the custom task UI and returns to the Worklist user portal by returning a forward marked as a ‘return’ forward, and specifying the stepDoneAction on that forward. More on this below.

- rejectLoanAction: Calls the helper functions to take the ‘Reject’ action on the task. This action also exits the custom task UI and returns to the user portal by returning a ‘return’ forward object specifying the stepDoneAction.

The following describes how to add new actions. You can follow the steps below and use the page flow action wizard to add all of the above actions (and then copy/paste the action method body code as given in the section ‘Implement Action Methods’). Or you can just copy and paste the complete code for the action declarations and methods as given in ‘Implement Action Methods’ below and skip the next step completely.

**Create an Action**

1. In the Page Flow Explorer, right-click Actions, new actions.

   The **New Action** dialog box appears (see **Figure 7-10**).
2. Create five new action and enter the details as shown in Table 7-1.

**Table 7-1 New Action Settings**

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Action Template</th>
<th>Form Beans</th>
<th>Forward To</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin</td>
<td>Basic...</td>
<td>&lt;none&gt;</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>viewAssetSummaryAction</td>
<td>Basic...</td>
<td>ManagerReviewForm</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>returnToManagerReviewAction</td>
<td>Basic...</td>
<td>AssetSummaryForm</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>approveLoanAction</td>
<td>Basic...</td>
<td>ManagerReviewForm</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>rejectLoanAction</td>
<td>Basic...</td>
<td>ManagerReviewForm</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

Next we need to implement a method body for the action methods we just defined.
The code for the all action methods is given below. Make sure you copy the action signature along with the @Jpf.Action annotation for each action method.

**Implement the Action Methods**

For each action method described in the above section:

If you didn’t create the action methods using the action wizard, you should:
- copy and paste the entire action method declaration and body.

If you did create the action methods using the action wizard you should:
- copy the method body code below (for a specific action method) and paste it inside the action method declaration you created for that action method (in the above section).
- make sure the annotations in your action method declaration match the annotations given in the code below for the named action method.

```java
/**
 * Initialize this controller, and call the super class helper to initialize stuff we get for free. This includes
 * task context, standard form beans for a task and action, and property editing support.
 */
@Jpf.Action(forwards = {
    @Jpf.Forward(name="success", path="GetManagerReview.jsp")
})
public Forward begin() throws Exception {
    // Initialize our base class helpers so we can use them throughout this controller
    beginActionHelper();
    // Create our ManagerReviewForm, and load it with property values given by our base class helpers
    _managerReviewForm = new ManagerReviewForm();
    _managerReviewForm.setName((String)getTaskPropertiesMap().get("Name").getValue());
    ```
_managerReviewForm.setSsn((String)getTaskPropertiesMap().get("SSN").getValue());

_managerReviewForm.setLoanAmount(((Long)getTaskPropertiesMap().get("LoanAmt").getValue()).intValue());

// Get the editable notes property, because we'll use this PropertyInstanceHolder to edit the notes property via Worklist-provided helpers
com.bea.wli.worklist.portal.PropertyInstanceHolder notesProp = getTaskEditablePropertiesMap().get("Notes");

_managerReviewForm.setNotesProp(notesProp);

return new Forward("success", _managerReviewForm);

/**
 * Forward to the assets sub form and display the assets we find for the loan applicant.
 */
@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
        path = "AssetSummary.jsp"
    ),
    useFormBean = "_managerReviewForm"
})

public Forward viewAssetSummaryAction(ManagerReviewForm form) {
    AssetSummaryForm assetSummaryForm = new AssetSummaryForm();
    assetSummaryForm.setName(form.getName());
    assetSummaryForm.setSsn(form.getSsn());
    assetSummaryForm.loadSummaryInfo(getSession());
    return new Forward("success", assetSummaryForm);
}
/**
 * Return to the main form after looking at assets.
 */
@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
        path = "GetManagerReview.jsp")
})
public Forward returnToManagerReviewAction(AssetSummaryForm form) {
    Forward forward = new Forward("success", _managerReviewForm);
    return forward;
}
/**
* Approve the loan, using the super class helpers. and the properties
* we
* stored in ManagerReviewForm. We
* specify the useFormBean attr to keep a single copy
* of ManagerReviewForm.
* NOTE: We could have designed this action to forward to an 'action
* props'
* page to collect the properties for the action (instead of
* putting
* fields directly on the main form. If we did want a separate
* page,
* we could call showStepActionActionHelper to prepare a
* TakeStepActionActionForm for us to obtain these properties
* from.
* This form is well suited to use with propertyEditor tags in the
* action props form.
* @see
TaskUIPageFlowController#showStepActionActionHelper(com.bea.wli.worklist.api.taskplan.StepAction)
* @see
TaskUIPageFlowController#takeStepActionActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm)
@see TaskUIPageFlowController#isPostActionInteractiveAssignment(java.lang.String)
@see TaskUIPageFlowController#takeStepActionAndClaimActionHelper(com.bea.wli .worklist.portal.TakeStepActionActionForm, java.lang.String)

```java
@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
        action = "stepDoneAction")
}, useFormBean="_managerReviewForm")
public Forward approveLoanAction(ManagerReviewForm form)
    throws Exception {
    // Build a map of the property values we'll pass for the action
    java.util.Map<String, String> propMap
        = new java.util.HashMap<String, String>();
    propMap.put("Notes", form.getNotesProp().getEditorValueAsString());
    propMap.put("CollateralAssets", form.getCollateralAssets());
    // Now take the action
    this.takeStepAction(getCurrentStep().getName(),
        "Approve", propMap);
    Forward forward = new Forward("success");
    return forward;
}
```

/*
 * Reject the loan, using the super class helpers. We
 * specify the useFormBean attr to keep a single copy
 * of ManagerReviewForm.
 * NOTE: We could have designed this action to forward to an 'action
 * props'
page to collect the properties for the action (instead of putting fields directly on the main form. If we did want a separate page, we could call showStepActionActionHelper to prepare a TakeStepActionActionForm for us to obtain these properties from. This form is well suited to use with propertyEditor tags in the action props form.

@see TaskUIPageFlowController#showStepActionActionHelper(com.bea.wli.worklist.api.taskplan.StepAction)
@see TaskUIPageFlowController#takeStepActionActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm)
@see TaskUIPageFlowController#isPostActionInteractiveAssignent(java.lang.String)
@see TaskUIPageFlowController#takeStepActionAndClaimActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm, java.lang.String)

@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
        action = "stepDoneAction")
}, useFormBean="_managerReviewForm")
public Forward rejectLoanAction(ManagerReviewForm form)
    throws Exception {
    // Build a map of the property values we'll pass for the action
    java.util.Map<String, String> propMap
        = new java.util.HashMap<String, String>();
    propMap.put("Notes", form.getNotesProp().getEditorValueAsString());
    // Now take the action
    this.takeStepAction(getCurrentStep().getName(),

"Reject",
    propMap);
    Forward forward = new Forward("success");
    return forward;
}

**Action Methods, Form Beans and the useFormBean Field**

Action methods can accept form beans, and forward to pages using form beans. When submitting a web form, and in the process of calling the action associated with the submit, the NetUI framework, by default, will create a new form bean instance (using the no-arg public constructor for the form bean class). This new bean instance is then populated via Java reflection with data from data binding tags in the submitted web page form.

This process has some limitations. For example, if your form bean contains transient, hidden information that is not represented in the web pages JSP tags, the form bean that actually gets passed to the action method (the bean that is created by the NetUI framework) will be missing this information.

To avoid the overhead and possible behavioral problems of creating new beans each time an action method is called, you can specify a useFormBean field on the @Jpf.Action annotation for an action method. This allows the controller to hold a single copy of the form bean in the page flow controller’s state, and the action method then just fetches the object from that state instead of creating a new form bean object.

We make use of the the useFormBean facility in the action method code given in the previous section. To make this code work, you need to define a member variable on the page flow controller to hold the form bean we’ll be passing around.

Add the following member variable to the top of your ManagerReview class:

```java
private ManagerReviewForm _managerReviewForm; // To preserve the form between requests.
```

If you haven’t already done so, make sure your action methods that take a ManagerReviewForm parameter include a useFormBean attribute in the @Jpf.Action annotation. For example, the @Jpf.Action annotation for the rejectLoanAction is:

```java
@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
               action = "stepDoneAction")
```
The text you need to add is highlighted in the above code.

**Worklist Property Editors and Actions**

Worklist provides some built-in support for editing properties in your custom task UI. It includes a JSP tag, default editors, and some helper methods in the base TaskUIPageFlowController. These facilities allow you to easily edit the following types of properties using out-of-box UI:

- Multi-line/mult-page text
- JavaBean/XMLBean objects

In addition, the property editor facility allows you to easily support editing properties using an inline editor (simple form field) as well as a stand-alone editor for the complex types mentioned above. This facility makes robust editing of properties a fairly simple matter. The manager review web page defined in this tutorial edits two properties; Notes and CollateralAssets. We edit the Notes property using the property editor facilities of Worklist, and the CollateralAssets property using simple NetUI data binding tags.

The property editor facility usage in this tutorial spans several constructs:

- a `<worklist:propertyEditor>` tag in the GetManagerReview.jsp page
- three actions in the ManagerReview page flow to handle forwarding out to the stand-alone text editor, and returning from that editor (for both the ‘Ok’ and ‘Cancel’ cases in that editor). These are, respectively, editNotesPropAction, okPropAction, cancelPropAction.

We define the following actions to handle task’s user property editor:

- **okPropAction** – The stand-alone editor (forwarded to in editNotesPropAction) returns to this action when you click 'Ok' to apply the edit. It returns on this action passing an EditorValueHolder holding the value that was created/edited in the editor. We pass this _editorValue in useFormBean to avoid creating a copy of this potentially large form bean.

- **cancelPropAction** – This is the action the stand-alone editor (launched from editNotesPropAction) calls when the user clicks Cancel in the editor.

- **editNotesPropAction** – This action handles an 'initiate stand-alone editor' call that comes from the GetManagerReview.jsp and the worklist propertyEditor tag. It (via editPropActionHelper) calculates the stand-alone editor's URI, and then forwards to that URI. This editor is a nested page flow, and returns to this controller (the caller) via
well-known return actions okPropAction, and cancelPropAction. We pass the ManagerReviewForm form bean to avoid it getting recreated in this call.

Using the steps described for adding the actions in Define Actions on the Page Flow section, add the following actions as shown in Table 7-2.

Table 7-2 Define Action on the Page Flow

<table>
<thead>
<tr>
<th>Action Name</th>
<th>Action Template</th>
<th>Form Bean</th>
<th>Forward To</th>
</tr>
</thead>
<tbody>
<tr>
<td>editNotesPropAction</td>
<td>Basic...</td>
<td>ManagerReviewForm</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>okPropAction</td>
<td>Basic...</td>
<td>com.bea.wli.datatype.EditorValueHolder</td>
<td>&lt;none&gt;</td>
</tr>
<tr>
<td>cancelPropAction</td>
<td>Basic...</td>
<td>&lt;none&gt;</td>
<td>&lt;none&gt;</td>
</tr>
</tbody>
</table>

Notes: For adding com.bea.wli.datatype.EditorValueHolder you will have to click Add button next to the form bean input field. It will open up a search window. Type EditorValueHolder and it should find this class. Click on the entry and press Ok.

Insert the following code for the three actions mentioned above into your ManagerReview page flow controller.

```java
private transient com.bea.wli.datatype.EditorValueHolder _editorValue;
// For efficiency
/**
 * This action handles an 'initiate stand-alone editor' call
 * that comes from the GetManagerReview.jsp and the worklist
 * propertyEditor tag. It (via editPropActionHelper) calculates
 * the stand-alone editor's URI, and then forwards to that URI.
 * This editor is a nested page flow, and returns to this
 * controller (the caller) via well-known return actions
 * okPropAction, and cancelPropAction. We pass the managerReviewForm
 * form bean to avoid it getting recreated in this call.
 */
@Jpf.Action(useFormBean="_managerReviewForm")
```
public Forward editNotesPropAction(ManagerReviewForm form) throws com.bea.wli.worklist.api.ManagementException, com.bea.wli.datatype.DataTypeException {
    // Get editable properties from the super class. Note
    // that we could also get these from the UpdateActionForm
    // contained in the super class. The UpdateActionForm is
    // maintained for us by our super
    // class, and contains the editable properties for the
    // task (these are represented as PropertyInstanceHolder)
    // General-purpose task UI can simply use the UpdateActionForm
    // as the form bean for their main page.
    com.bea.wli.worklist.portal.PropertyInstanceHolder[] properties =
        getTaskEditablePropertiesMap().
            values().toArray(new
    com.bea.wli.worklist.portal.PropertyInstanceHolder[0]);
    // NOTE: We might store attrs off the propertyEditor tag
    // here (e.g. hostPage) that would help us to
    // navigate back to an appropriate page when the edit
    // is completed (via okPropAction) or aborted (via
    // cancelPropAction

    // This begins the edit on the property we selected
    // in the JSP page (and the name is set into the HTTP
    // request coming in on this method.
    Forward forward = editPropActionHelper(properties);
    return forward;
}
/**
 * The stand-alone editor (forwarded to in editNotesPropAction)
 * returns to this action when you click 'Ok' to apply the edit.
 * It returns on this action passing an EditorValueHolder holding
* the value that was created/edited in the editor. We pass
* this _editorValue in useFormBean to avoid creating a copy
* of this potentially large form bean.
*/
@Jpf.Action(loginRequired = true,
    forwards = {
        @Jpf.Forward(name = "backToManagerReview",
                     path = "GetManagerReview.jsp")
    },
    useFormBean = "_editorValue")
protected Forward okPropAction(com.bea.wli.datatype.EditorValueHolder value)
    throws Exception {
        okPropActionHelper(value);
        return new Forward("backToManagerReview", _managerReviewForm);
    }
/**
* This is the action the stand-alone editor (launched from
* editNotesPropAction) calls when the user clicks Cancel in
* the editor.
* @return
* @throws Exception
*/
@Jpf.Action(loginRequired = true,
    forwards = {
        @Jpf.Forward(name = "backToManagerReview",
                     path = "GetManagerReview.jsp")
    })
protected Forward cancelPropAction()
    throws Exception {
        cancelPropActionHelper();
    }
return new Forward("backToManagerReview", _managerReviewForm);
}

Final Code for Page Flow
After completing the above mentioned steps the code for the Page Flow will be as follows:

```java
package manager;
import javax.servlet.http.HttpSession;
import org.apache.beehive.netui.pageflow.Forward;
import org.apache.beehive.netui.pageflow.annotations.Jpf;
import com.bea.wli.worklist.portal.PropertyInstanceHolder;
import com.bea.wli.worklist.portal.TaskUIPageFlowController;
@Jpf.Controller(nested = true)
public class ManagerReview extends TaskUIPageFlowController {
    private static final long serialVersionUID = -1579985639L;
    private ManagerReviewForm _managerReviewForm; // To preserve the form between requests.

    @Jpf.Action(forwards = { @Jpf.Forward(name = "done", returnAction = "managerDone") })
    protected Forward done() {
        return new Forward("done");
    }

    /**
     * Initialize this controller, and call the super class
     * helper to initialize stuff we get for free. This includes
     * task context, standard form beans for a task and action,
     * and property editing support.
     */
    @Jpf.Action(forwards = {
        @Jpf.Forward(name="success", path="GetManagerReview.jsp")
    })
    public Forward begin() throws Exception {
```
beginActionHelper();

// Create our ManagerReviewForm, and load it with property
// values given by our base class helpers
_managerReviewForm = new ManagerReviewForm();
_managerReviewForm.setName((String)getTaskPropertiesMap().
    get("Name").getValue());
_managerReviewForm.setSsn((String)getTaskPropertiesMap().
    get("SSN").getValue());
_managerReviewForm.setLoanAmount((Long)getTaskPropertiesMap().
    get("LoanAmt").getValue()).intValue();

// Get the editable notes property, because we'll
// use this PropertyInstanceHolder to edit the notes
// property via Worklist-provided helpers
PropertyInstanceHolder notesProp =
    getTaskEditablePropertiesMap().
    get("Notes");
_managerReviewForm.setNotesProp(notesProp);
return new Forward("success", _managerReviewForm);

/**
 * Forward to the assets sub form and display the assets
 * we find for the loan applicant.
 */
@Jpf.Action(forwards = {

7-32 Tutorial: Building a Worklist Application
public Forward viewAssetSummaryAction(ManagerReviewForm form) {
    AssetSummaryForm assetSummaryForm = new AssetSummaryForm();
    assetSummaryForm.setName(form.getName());
    assetSummaryForm.setSsn(form.getSsn());
    assetSummaryForm.loadSummaryInfo(getSession());

    return new Forward("success", assetSummaryForm);
}

/**
 * Return to the main form after looking at assets.
 */
@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
                 path = "GetManagerReview.jsp")
})
public Forward returnToManagerReviewAction(AssetSummaryForm form) {
    Forward forward = new Forward("success", _managerReviewForm);
    return forward;
}

/**
 * Approve the loan, using the super class helpers. and the properties we
 * stored in ManagerReviewForm. We
 * specify the useFormBean attr to keep a single copy
 * of ManagerReviewForm.
 */
NOTE: We could have designed this action to forward to an 'action props' page to collect the properties for the action (instead of putting fields directly on the main form. If we did want a separate page, we could call showStepActionActionHelper to prepare a TakeStepActionActionForm for us to obtain these properties from. This form is well suited to use with propertyEditor tags in the action props form.

@see TaskUIPageFlowController#showStepActionActionHelper(com.bea.wli.worklist.api.tasktype.StepAction)
@see TaskUIPageFlowController#takeStepActionActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm)
@see TaskUIPageFlowController#isPostActionInteractiveAssignment(java.lang.String)
@see TaskUIPageFlowController#takeStepActionAndClaimActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm, java.lang.String)

@Jpf.Action(forwards = {
        @Jpf.Forward(name = "success",
                action = "stepDoneAction")
    }, useFormBean="_managerReviewForm")
public Forward approveLoanAction(ManagerReviewForm form)
    throws Exception {
    // Build a map of the property values we'll pass for the action
    java.util.Map<String, String> propMap
        = new java.util.HashMap<String, String>();
    propMap.put("Notes", form.getNotesProp().getEditorValueAsString());
    propMap.put("CollateralAssets", form.getCollateralAssets());
// Now take the action
this.takeStepAction(getCurrentStep().getName(),
    "Approve",
    propMap);
Forward forward = new Forward("success");
return forward;
}

/**
 * Reject the loan, using the super class helpers. We
 * specify the useFormBean attr to keep a single copy
 * of ManagerReviewForm.
 * NOTE: We could have designed this action to forward to an 'action
 * props'
 * page to collect the properties for the action (instead of
 * putting
 * fields directly on the main form. If we did want a separate
 * page,
 * we could call showStepActionActionHelper to prepare a
 * TakeStepActionActionForm for us to obtain these properties
 * from.
 * This form is well suited to use with propertyEditor tags in the
 * action props form.
 * @see TaskUIPageFlowController#showStepActionActionHelper(com.bea.wli.worklist.api.tasktype.StepAction)
 * @see TaskUIPageFlowController#takeStepActionActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm)
 * @see TaskUIPageFlowController#isPostActionInteractiveAssignment(java.lang.String)
 * @see TaskUIPageFlowController#takeStepActionAndClaimActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm, java.lang.String)
public Forward rejectLoanAction(ManagerReviewForm form)
    throws Exception {
    // Build a map of the property values we'll pass for the action
    java.util.Map<String, String> propMap = new java.util.HashMap<String, String>();
    form.getNotesProp().getEditorValueAsString();
    propMap.put("Notes", propMap.put("Notes", form.getNotesProp().getEditorValueAsString()));
    // Now take the action
    this.takeStepAction(getCurrentStep().getName(), "Reject", propMap);
    Forward forward = new Forward("success");
    return forward;
}

/**
 * Callback that is invoked when this controller instance is created.
 */
@Override
protected void onCreate() {
}

/**
 * Callback that is invoked when this controller instance is destroyed.
 */
@override

protected void onDestroy(HttpServletRequest session) {
}

private transient com.bea.wli.datatype.EditorValueHolder _editorValue; // For efficiency

/**
 * This action handles an 'initiate stand-alone editor' call
 * that comes from the GetManagerReview.jsp and the worklist
 * propertyEditor tag. It (via editPropActionHelper) calculates
 * the stand-alone editor's URI, and then forwards to that URI.
 * This editor is a nested page flow, and returns to this
 * controller (the caller) via well-known return actions
 * okPropAction, and cancelPropAction. We pass the managerReviewForm
 * form bean to avoid it getting recreated in this call.
 */
@Jpf.Action(useFormBean="_managerReviewForm")
public Forward editNotesPropAction(ManagerReviewForm form)
    throws com.bea.wli.worklist.api.ManagementException,
            com.bea.wli.datatype.DataTypeException {
    // Get editable properties from the super class. Note
    // that we could also get these from the UpdateActionForm
    // contained in the super class. The UpdateActionForm is
    // maintained for us by our super
    // class, and contains the editable properties for the
    // task (these are represented as PropertyInstanceHolder)
    // General-purpose task UI can simply use the UpdateActionForm
    // we could call showStepActionActionHelper to prepare a
    * TakeStepActionActionForm for us to obtain these properties
    from.
    * This form is well suited to use with propertyEditor tags in the
* action props form.

* @see TaskUIPageFlowController#showStepActionActionHelper(com.bea.wli.worklist.api.tasktype.StepAction)

* @see TaskUIPageFlowController#takeStepActionActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm)

* @see TaskUIPageFlowController#isPostActionInteractiveAssignment(java.lang.String)

* @see TaskUIPageFlowController#takeStepActionAndClaimActionHelper(com.bea.wli.worklist.portal.TakeStepActionActionForm, java.lang.String)

*/

@Jpf.Action(forwards = {
   @Jpf.Forward(name = "success",
       action = "stepDoneAction")
}, useFormBean="_managerReviewForm")

public Forward rejectLoanAction(ManagerReviewForm form)
   throws Exception {
   // Build a map of the property values we'll pass for the action
   java.util.Map<String, String> propMap = new java.util.HashMap<String, String>();
   propMap.put("Notes", form.getNotesProp().getEditorValueAsString());
   // Now take the action
   this.takeStepAction(getCurrentStep().getName(), "Reject",
       propMap);
   Forward forward = new Forward("success");
   return forward;
}

/**
* Callback that is invoked when this controller instance is created.

@override
protected void onCreate() {
}

/**
 * Callback that is invoked when this controller instance is destroyed.
 */
@override
@override
protected void onDestroy(HttpSession session) {
}

private transient com.bea.wli.datatype.EditorValueHolder _editorValue; // For efficiency

/**
 * This action handles an 'initiate stand-alone editor' call
 * that comes from the GetManagerReview.jsp and the worklist
 * propertyEditor tag. It (via editPropActionHelper) calculates
 * the stand-alone editor's URI, and then forwards to that URI.
 * This editor is a nested page flow, and returns to this
 * controller (the caller) via well-known return actions
 * okPropAction, and cancelPropAction. We pass the managerReviewForm
 * form bean to avoid it getting recreated in this call.
 */
@Jpf.Action(useFormBean="_managerReviewForm")
public Forward editNotesPropAction(ManagerReviewForm form)
    throws com.bea.wli.worklist.api.ManagementException,
            com.bea.wli.datatype.DataTypeException {
    // Get editable properties from the super class. Note
// that we could also get these from the UpdateActionForm
// contained in the super class. The UpdateActionForm is
// maintained for us by our super
// class, and contains the editable properties for the
// task (these are represented as PropertyInstanceHolder)
// General-purpose task UI can simply use the UpdateActionForm

public String getSsn() { return _ssn; }
public void setSsn(String ssn) { _ssn = ssn; }
public PropertyInstanceHolder getNotesProp()
{ return _notesProp; }
public void setNotesProp(PropertyInstanceHolder notesProp)
{ _notesProp = notesProp; }
public String getCollateralAssets() { return _collateralAssets; }
public void setCollateralAssets(String collateralAssets) {
    _collateralAssets = collateralAssets;
}

@Jpf.FormBean
public static class AssetSummaryForm implements java.io.Serializable
{

private static final long serialVersionUID = 1517513921L;

private java.util.SortedSet<AssetForm> _assets;
private int _creditScore;
private String _name;
private String _ssn;

public AssetSummaryForm()
{ 
    _assets = new java.util.TreeSet<AssetForm>();
}
public String getName() { return _name; }
public void setName(String name) { _name = name; }
public String getSsn() { return _ssn; }
public void setSsn(String ssn) { _ssn = ssn; }
public java.util.SortedSet<AssetForm> getAssets() { return _assets; }

// NOTE: No setter for assets property. We’ll load this internally.
public int getCreditScore() { return _creditScore; }
// NOTE: No setter for creditScore, we’ll load this internally.
public void loadSummaryInfo(HttpSession session) {
    loadCreditScore(session);
    loadAssets(session);
}

public int getTotalActualAssetValue() {
    int total = 0;
    for (AssetForm asset: _assets) {
        total = asset.getActualValue();
    }
    return total;
}

protected void loadCreditScore(HttpSession session) {
    // Load the credit scores as properties
    String resourceName = "/creditRatings/creditRatings.properties";
    java.util.Properties props =
    loadProperties(resourceName, session);
    _creditScore = getIntProperty(props, _name);
}

protected void loadAssets(HttpSession session) {
    // Load the assets as properties
    String resourceName = "/assets/" + _name + ".properties";
    java.util.Properties props =
loadProperties(resourceName, session);
String assetList = props.getProperty("assetList");
if (assetList != null) {
    java.util.StringTokenizer st = new
    java.util.StringTokenizer(assetList, ",");
    while (st.hasMoreTokens()) {
        String assetName = st.nextToken().trim();
        AssetForm asset = new AssetForm();
        asset.setName(assetName);
        int value =
            getIntProperty(props, assetName + "." + "value");
        asset.setValue(value);
        int amountOwed =
            getIntProperty(props, assetName + "." + "amountOwed");
        asset.setAmountOwed(amountOwed);
        _assets.add(asset);
    }
}

protected java.util.Properties
loadProperties(String resourceName, HttpSession session) {
    // Load the resources as properties
    java.io.InputStream is = null;
    try {
        is = session.getServletContext().
            getResourceAsStream(resourceName);
        java.util.Properties props = new java.util.Properties();
        if (is != null) {
            props.load(is);
        }
    }
return props;
} catch (Exception e) {
   // TODO: Better handling
   e.printStackTrace();
} finally {
   if (is != null) {
      try { is.close(); } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
return new java.util.Properties();

    private int getIntProperty(java.util.Properties props, String key) {
      String value = props.getProperty(key);
      if (value == null) {
         return 0;
      }
      return Integer.valueOf(value);
    }

@Jpf.FormBean
public static class AssetForm implements java.io.Serializable, Comparable {
private static final long serialVersionUID = 1491696939L;

    private String _name;
    private int _value;
    private int _amountOwed;

    public String getName() {
      return _name;
    }

    public void setName(String name) {
      _name = name;
    }

    public int getValue() {
      return _value;
    }

    public void setValue(int value) {
      _value = value;
    }

    public int getAmountOwed() {
      return _amountOwed;
    }

    public void setAmountOwed(int owed) {
      _amountOwed = owed;
    }

    @Override
    public boolean equals(Object obj) {
      if (this == obj) {
        return true;
      }
      if (obj == null || getClass() != obj.getClass()) {
        return false;
      }
      AssetForm other = (AssetForm) obj;
      return _name.equals(other._name) && _value == other._value && _amountOwed == other._amountOwed;
    }

    @Override
    public int hashCode() {
      int result = _name != null ? _name.hashCode() : 0;
      result = 31 * result + _value;
      result = 31 * result + _amountOwed;
      return result;
    }

    @Override
    public int compareTo(Comparable o) {
      AssetForm other = (AssetForm) o;
      if (_value != other._value) {
        return _value - other._value;
      }
      return _name.compareTo(other._name);
    }

    public Object clone() {
      AssetForm clone = (AssetForm) super.clone();
      clone._name = _name;
      clone._value = _value;
      clone._amountOwed = _amountOwed;
      return clone;
    }

    public static void main(String[] args) {
      AssetForm assetForm = new AssetForm();
      assetForm.setName("Apple");
      assetForm.setValue(100);
      assetForm.setAmountOwed(20);
      System.out.println("Name: ", assetForm.getName());
      System.out.println("Value: ", assetForm.getValue());
      System.out.println("Owed: ", assetForm.getAmountOwed());
    }
  
  
}
public int getAmountOwed() {
    return _amountOwed;
}

public void setAmountOwed(int lienValue) {
    _amountOwed = lienValue;
}

public String getName() {
    return _name;
}

public void setName(String name) {
    _name = name;
}

public int getValue() {
    return _value;
}

public void setValue(int value) {
    _value = value;
}

public int getActualValue() {
    return _value - _amountOwed;
}

public int compareTo(Object o) {
    if (!(o instanceof AssetForm)) {
        return 0;
    }
    AssetForm other = (AssetForm)o;
    int otherActualValue = other._value - other._amountOwed;
    return otherActualValue - getActualValue();
Define JSP Pages

According to the screen mockups in Define Web Page Mock-Up and Flow, define two JSP pages for the custom task UI using Beehive NetUI data binding JSP tags to render web forms that can read data from and write data into form beans. The use of these tags greatly simplifies the process of writing a data-driven JSP page.

Create JSP Files

The page flow perspective in Workshop give us a starting point for defining the correct pages. With the actions we defined in previous sections, your ManagerReview page flow controller should show two grayed out JSP pages under the ‘Pages’ node in the page flow explorer:

- GetManagerReview.jsp
- AssetSummary.jsp

To create the JSP Files

1. In the Page Flow Explorer, right-click Pages.
2. Select GetManagerReview.jsp → Create to create the new JSP file
3. Repeat the above steps for AssetSummary.jsp.

The default JSP code is as follows:

```jsp
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>

<netui:html>
  <head>
    <netui:base/>
  </head>
```
Create the Form and Associate it with an Action

In this step, create a `<netui:form>` element to hold all our JSP data items, then associate that form with an action from our ManagerReview page flow controller. This will associate the form with the form bean referenced in the action method. This association will bind the data from our form bean to the data items we’ll add to the JSP form.

The general process for adding a NetUI form to a JSP is as follows:

1. If the JSP Design Palette is not visible, go to **Window→Show View→JSP Design Palette**. The JSP Design Palette appears (see *Figure 7-11*).
2. In the JSP Design Palette, expand NetUI and select form.

3. Drag the form into your JSP source editor and drop it inside the <netui:body/> tag.

   When you drop it, the form element is added, something like this:
   
   `<netui:form action=""></netui:form>`

   Associate the form with an action on your page flow controller. For this tutorial, you'll associate the form element with an action on our ManagerReview page flow controller that takes a form bean as a parameter. This association is very important, as it establishes the action method that will be called when the form is submitted and the Java type of the form bean to associate with the form. The associated form bean then becomes accessible to the NetUI tags in the JSP code by using the value and dataSource attributes of those tags. These attributes refer to properties on the form bean via JSP expressions like this:

   `actionForm.<property on form bean>`

   and the form bean defines a pair of methods of the form:

   `<Java type for property> get<Property name>()`  
   `void set<Property name>({<Java type for property> value})`
We’ll show examples of this for the individual web pages we define.

**GetManagerReview.jsp**

4. Double-click **GetManagerReview.jsp** and open it in the source editor.

5. Drag the **form** from **NetUI** to the **GetManagerReview.jsp** and set the action for that form to **approveLoanAction**. The fact that approveLoanAction takes a form bean parameter of ManagerReviewForm type generates the association, within the form tag in GetManagerReview.jsp only, that:

   `actionForm.<property> -> Call method ManagerReviewForm.get<property>`

6. The instance of ManagerReviewForm that is used to make this call is the instance passed in the Forward object that forwarded to this page. In the ManagerReview.java page flow code, the following happens from these actions:

   – `begin()Add Data Binding Fields`
   – `returnToManagerReviewAction()`
   – `okPropAction()`
   – `cancelPropAction()`

   All of these actions should return a Forward containing a ManagerReviewForm instance.

   The JSP Code should be as shown below:

   ```jsp
   <%@ page language="java" contentType="text/html;charset=UTF-8"%>
   <%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
   <%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
   <%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
   <netui:html>
   <head>
     <netui:base/>
   </head>
   <netui:body>
   <netui:form action="approveLoanAction"></netui:form>
   </netui:body>
   ```
AssetSummary.JSP

We’ll create a <netui:form> and set the action for that form to be returnToManagerReviewAction. Remember that returnToManagerReviewAction looks like this:

```java
@Jpf.Action(forwards = {
    @Jpf.Forward(name = "success",
                 path = "GetManagerReview.jsp"),
    @Jpf.Forward(name = "success2",
                 path = "GetManagerReview2.jsp")
})

public Forward returnToManagerReviewAction(AssetSummaryForm form) {
    Forward forward = new Forward("success", _managerReviewForm);
    return forward;
}
```

The fact that returnToManagerReviewAction takes a form bean parameter of AssetSummaryForm type generates the association, within the form tag in AssetSummary.jsp only, that:

`actionForm.<property> -> Call method AssetSummaryForm.get<property>`

The instance of AssetSummaryForm that is used to make this call is the instance passed in the Forward object that forwarded to this page. In our ManagerReview.java page flow code, the AssetSummary.jsp is reached from these actions:

The JSP Code should be as shown below:

```html
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<netui:html>
    <head>
```
Add Data-binding Fields

NetUI data binding JSP tags automate the work needed to fetch data out of a form bean for display in a JSP page, and to set data into a form bean as a result of submitting a `<netui:form>` tag. All NetUI data binding tags have an attribute that associates them with a property on a form bean. In some tags (e.g. `<netui:label>`) the attribute is named ‘value’. On others (e.g. `<netui-data:repeater>`) the attribute is named ‘dataSource’. These attributes use a different syntax for defining the property expression. If the attribute is dataSource, the syntax is:

```
actionForm.<property>
```

If the attribute is anything else (e.g. value on `<netui:label>`) the syntax is:

```
${actionForm.<property>}
```

We use actionForm references in the following JSP code to bind properties from our ManagerReviewForm to the JSP page. You can drag and drop the appropriate tags from the JSP Designer palette to the JSP code to arrive at these results. Note that some of the tags come from the NetUI menu, and others from the NetUI-Data menu (e.g. repeater). The final code for our JSP files are given in the sections below:

**Final Code GetManagerReview.jsp**

```jsp
<%@ page language="java" contentType="text/html;charset=UTF-8"

<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>

<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
```
<netui:html>
  <head>
    <netui:base/>
  </head>
  <netui:body>
    <netui:form action="approveLoanAction">
      <table>
        <tr>
          <td><netui:label value="Customer Name:"/></td>
          <td><netui:label value="${actionForm.name}"/></td>
        </tr>
        <tr>
          <td><netui:label value="SSN:"/></td>
          <td><netui:label value="${actionForm.ssn}"/></td>
        </tr>
        <tr>
          <td><netui:label value="Loan Amount:"/></td>
          <td><netui:label value="${actionForm.loanAmount}"/></td>
        </tr>
        <tr>
          <td colspan="2">
            <!-- We'll edit this property using a plain-old NetUI actionForm binding (Note the netui:textBox tag) -->
            <netui:label value="Collateral Assets:"/>
            <netui:textBox dataSource="actionForm.collateralAssets"/>
          </td>
        </tr>
      </table>
    </netui:form>
  </netui:body>
</netui:html>
Notice that we don’t have any tags to handle the Notes property. We cover this separately here because we’ll use a custom Worklist tag called propertyEditor to allow us to use the property editing framework offered by Worklist. Insert the following code before the last <tr> element in the table above (the one that holds the CollateralAssets property elements)

<tr><td colspan="2">
<netui:label value="Reason for Action:"/>
<worklist:propertyEditor dataSource="actionForm.notesProp" propName="Notes" readOnly="false"/>
</td></tr>
To make the `<worklist:propertyEditor>` tag reference legal, we must define the worklist prefix to map to the correct URI for the Worklist tags. Add this to the end of the taglib statements at the top of the JSP file:

```jsp
<%@taglib uri="http://bea.com/wli/worklist/tags-worklist-1.0" prefix="worklist"%>
```

The `propertyEditor` tag is bound to `actionForm.notesProp` which is of type `PropertyInstanceHolder`. This binding allows the `propertyEditor` tag to retrieve an editable property value for the property, determine its property type (one of the Worklist-defined types), find the registered stand-alone editor for that type, and pass the editable value to the stand-alone editor.

**Note:** The `propertyEditor` tag refers to our `editNotesPropAction`. This action will be called when launching the stand-alone editor for the property. It is not apparent here, but our code in the ManagerReview page flow controller also includes two `return` action methods that allow the stand-alone editor to return to the calling page flow when its Ok and Cancel buttons are clicked. These actions are `okPropAction`, and `cancelPropAction`, respectively.

**Final Code Asset Summary.jsp**

The final code for `Asset Summary.jsp` is as follows:

```jsp
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
```
<netui:html>
  <head>
    <netui:base/>
  </head>
  <netui:body>
    <netui:form action="returnToManagerReviewAction">
      <table>
        <tr>
          <td colspan="2"><netui:label value="Assets for ${actionForm.name}"/></td>
        </tr>
        <tr>
          <netui-data:repeater dataSource="actionForm.assets">
            <netui-data:repeaterHeader>
              <table border="1">
                <tr>
                  <td>Name</td>
                  <td>Value</td>
                  <td>Amount Owed</td>
                  <td>Actual Value</td>
                </tr>
              </table>
            </netui-data:repeaterHeader>
            <netui-data:repeaterItem>
              <tr>
                <td><netui:label value="${container.item.name}"/></td>
                <td><netui:label value="${container.item.value}"/></td>
                <td><netui:label value="${container.item.owes}"/></td>
                <td><netui:label value="${container.item.actual_value}"/></td>
              </tr>
            </netui-data:repeaterItem>
          </netui-data:repeater>
        </tr>
      </table>
    </netui:form>
  </netui:body>
</netui:html>
Add Command Links and Buttons
Add <netui:button> elements to navigate between forms and to take actions on the task they represent.

GetManagerReview.jsp
Insert the following code before/above the <tr> element containing the <worklist:propertyEditor> tag. This code renders an action button in our GetManagerReview page that when clicked, will call the viewAssetSummary action, and forward the user to the AssetSummary page.

<tr>
    <td colspan="2">
        <netui:button value="Call viewAssetSummary action"/>
    </td>
</tr>
<netui:button value="View Asset Summary"
action="viewAssetSummaryAction"/>
</td>
</tr>

and then insert this code before the ending </table> tag
<tr>
<td colspan="2">
<netui:button value="Approve"
action="approveLoanAction"/>
<p/>
<netui:button value="Reject" action="rejectLoanAction"/>
</td>
</tr>

This renders buttons that allow the Loan Manager to take the Approve and Reject actions on the
task (via the approveLoanAction and rejectLoanAction action methods on the page flow).

Asset Summary.jsp
Insert the following code before the end table tag (</table>):
<tr>
<td colspan="2"><netui:button value="Back"/>
</tr>

This renders a 'Back' button to take the Loan Manager back to the Manager Review page. Note
that there is no action attribute here. In this case, the action from the form
(returnToManagerReviewAction) is taken.

**Final JSP Code**
The final jsp code for GetManagerReview.jsp is shown below:
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>
<%@taglib uri="http://bea.com/wli/worklist/tags-worklist-1.0" prefix="worklist"%>
<netui:html>
  <head>
    <netui:base/>
  </head>
  <netui:body>
    <netui:form action="approveLoanAction">
      <table>
        <tr>
          <td><netui:label value="Customer Name:"/></td>
          <td><netui:label value="${actionForm.name}"/></td>
        </tr>
        <tr>
          <td><netui:label value="SSN:"/></td>
          <td><netui:label value="${actionForm.ssn}"/></td>
        </tr>
        <tr>
          <td><netui:label value="Loan Amount:"/></td>
          <td><netui:label value="${actionForm.loanAmount}"/></td>
        </tr>
      </table>
    </netui:form>
  </netui:body>
</netui:html>
<table>
<thead>
<tr>
<th>Reason for Action:</th>
<th>Notes</th>
</tr>
</thead>
</table>

**NOTE:** We set the `hostPage` attr to facilitate navigating back to this page after editing a property.

**NOTE:** We need an action defined on the controller that has the name given in the `actionName` attr.
The final JSP code for AssetSummary.jsp is shown below:

```jsp
<%@ page language="java" contentType="text/html;charset=UTF-8"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-html-1.0" prefix="netui"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-databinding-1.0" prefix="netui-data"%>
<%@taglib uri="http://beehive.apache.org/netui/tags-template-1.0" prefix="netui-template"%>

<netui:html>
  <head>
    <!-- We'll edit this property using a plain-old NetUI actionForm binding (Note the netui:textBox tag) -->
    <netui:label value="Collateral Assets:"/>
    <netui:textBox dataSource="actionForm.collateralAssets"/>
    <netui:button value="Approve" action="approveLoanAction"/>
    <netui:button value="Reject" action="rejectLoanAction"/>
  </head>
  <tr>
    <td colspan="2">
    </tr>
  </table>
</netui:form>
</netui:body>
</netui:html>
```
<netui:form action="returnToManagerReviewAction">
  <table>
    <tr>
      <td colspan="2"><netui:label value="Assets for ${actionForm.name}"/></td>
    </tr>
    <tr>
      <netui-data:repeater dataSource="actionForm.assets">
        <netui-data:repeaterHeader>
          <table border="1">
            <tr>
              <td>Name</td>
              <td>Value</td>
              <td>Amount Owed</td>
              <td>Actual Value</td>
            </tr>
          </table>
        </netui-data:repeaterHeader>
        <netui-data:repeaterItem>
          <tr>
            <td><netui:label value="${container.item.name}"/></td>
            <td><netui:label value="${container.item.value}"/></td>
            <td><netui:label value="${container.item.amountOwed}"/></td>
            <td><netui:label value="${container.item.amountOwed}"/></td>
          </tr>
        </netui-data:repeaterItem>
      </netui-data:repeater>
    </table>
  </netui:form>
Register the Custom UI

After designing the custom task UI for the Manager Review Pending step of the Loan Approval task plan, you need to register the custom task UI to be applied under those circumstances, by adding mapping entries to a registry file located in our LoanWeb web project, at the following location:

LoanWeb/WebContent/WEB-INF/task-ui-registry.xml

This XML file is associated with the schema for the Worklist Task UI Registry, and this schema is registered with Workshop. You have to edit this file in Workshop. Open the file, and you’ll
see an editor with Design and Source tabs. In the source tab, the initial contents should look something like this:

```xml
<task-ui-registry xmlns="http://www.bea.com/wli/worklist/taskuiregistry">
</task-ui-registry>
```

In the design tab, you can right-click any node in the tree view to act on it. You can delete nodes, and add children to nodes.

To register our custom task UI, we need the following information:

- **Task Plan ID** – The ID of the task plan for which the custom task UI applies, in external format (e.g. `/<path>/<task plan name>:<version>`)  
- **Step Name** – Required only when registering the custom task UI for a specific step. In this case, give the name of the step as it is defined in the task plan with the ID given above.  
- **Custom Task UI URI** – The web URI of the page flow controller that will control the UI to be applied to this task plan (and possibly step if Step Name is given)

For this tutorial, the required information is:

- **Task Plan ID** - `/Loans/Loan Approval:1.0`  
- **Step Name** – `Manager Review Pending`  
- **Custom Task UI URI** - `/manager/ManagerReview.jpf`  

The URI ends with `.jpf`, even though our page flow controller file is really `ManagerReview.java`.  
This is needed to allow servlet filters in the LoanApp web application to fire correctly.  
To finish editing `task-ui-registry.xml` first switch to the Package Explorer view and then do the following in the design tab of the editor:

1. In the Package Explorer, expand **LoanWeb** and go to `/WebContent/WEB-INF/`, and select `task-ui-registry.xml`.  

2. Right-click `task-ui-registry.xml`, and select **Add Child → step-override**. This adds a new step-override element under the root element (see Figure 7-12).
3. Expand the newly added step-override element. You’ll see three elements under it called, respectively, task-type-id, step-name, and custom-task-ui-uri. These all are set to default values.

4. For each of these three elements, click on the right-hand value column, and replace the default values with the l value discussed above (see Figure 7-13).

**Figure 7-13 Edited XML**

---

**Deploy the Custom Task UI**

1. On the Package Explorer pane, select and right-click on Loan_Web.

2. Click Run As→Run On Server.

3. In the Define a New Server dialog box, accept the default settings and click Next.
4. Browse and select the myworklist domain, which you created using the Configuration Wizard. It is located at \user_projects\domains\myworklist.

5. Click Finish.

**Validate the Custom UI**

To validate the Custom Task UI we use the following scenario:

- “Assigned loanOfficer Forwards Loan to his Manager”
- “Assigned loanManager Rejects the Loan.”

**Configure users and groups**

See Configure Users and Groups for Loan Processing.

**Create a Loan Approval Task**

1. Log in to the **Loan_Web** project using the following credentials:

   **Username:** weblogic
   
   **Password:** weblogic

2. The Home page is displayed with the portlets for the Inbox of overdue, upcoming, and assigned tasks, along with the portlet that allows you to create a new task.

3. Click the `/Loan/loan_approval 1.0` option in the Create Task portlet. The Create New Task page is displayed.

4. Enter the name **Loan for John Smith** as the Task Name.

5. Enter the following information:

   - Name- John Smith
   - LoanAmt - 20000
   - SSN- 111-11-1111

6. Click Create Task. The task is created and shows up in the Upcoming Tasks portlet on the home page.

7. Click Logout to close and log out as weblogic from the Worklist User Portal.
Assigned loanOfficer Forwards Loan to his Manager

1. Log in to the Loan_Web project using the following credentials:
   - **Username:** John
   - **Password:** password

2. Click **Loan for John Smith** in the Upcoming Tasks portlet. This will display the Task Work page with the task details, and the Action options available for user John (Figure 7-14).

   Figure 7-14  Task Detail Information

3. Select **Request Manager Review** in the Actions section to forward the request to the loan managers group for approval, and click **Next**.

4. In the Key Action Properties of the refreshed web page that appears, enter the String **Good Guy**.

5. Click **Submit** to complete the task.

6. As the loan has been send to the manager for approval, the task instance will no longer appear in John’s Inbox.
7. Logout as user John from the Worklist User Portal.

**Assigned loanManager Rejctes the Loan.**

Before creating a task instance for the new task plan, log into the Loan_web project using the following credentials:

**Username:** Mary

**Password:** password

Mary will see the ‘Loan for John Smith’ task in her Inbox. Click this task, the resulting page is the custom task UI page flow we defined above (see Figure 7-15).

**Figure 7-15 View Asset Summary**

![Asset Summary](image)

1. From the Manager Review Page, click **View Asset Summary**.

This displays John Smith’s asset as shown in Figure 7-16.

**Figure 7-16 Asset Summary**

![Asset Summary](image)
This table lists John Smith’s assets, and their actual value (in descending order). Mary looks at this information, and realizes John Smith has only $10,000 in assets, and low credit score (100). Mary decides to reject this loan by performing the following steps:

2. Click **Back** in the Asset Summary Page, to return to the Manager Review form.

3. Enter Insufficient assets and low credit score in Reason for Action.

4. Click **Reject** as shown in Figure 7-17, and the loan application is rejected by the manager. This puts the task in an aborted state and removes it from Mary’s Upcoming task portlets.

**Figure 7-17 Loan Reject**