



BEA WebLogic Integration™

Tutorial: Building a Worklist Application

Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, Jolt, Tuxedo, and WebLogic are registered trademarks of BEA Systems, Inc. BEA Builder, BEA Campaign Manager for WebLogic, BEA eLink, BEA Liquid Data for WebLogic, BEA Manager, BEA WebLogic Commerce Server, BEA WebLogic Enterprise, BEA WebLogic Enterprise Platform, BEA WebLogic Enterprise Security, BEA WebLogic Express, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Platform, BEA WebLogic Portal, BEA WebLogic Server, BEA WebLogic Workshop and How Business Becomes E-Business are trademarks of BEA Systems, Inc.

All other trademarks are the property of their respective companies.

Contents

About This Document

What You Need to Know	v
e-docs Web Site	v
How to Print the Document	vi
Related Information	vi
Contact Us!	vi
Documentation Conventions	vii

1. Tutorial: Building a Worklist Application

Tutorial Overview	1-1
Steps in This Tutorial	1-4

2. Step 1. Set Up Your Environment

Before You Begin	2-1
Create a New WebLogic Integration Domain	2-1
Configure the Users and Groups	2-3
Create a Business Calendar	2-6

3. Step 2. Create Your Application

Start WebLogic Workshop	3-1
Create a WebLogic Workshop Application	3-2
Create the Tutorial Schemas	3-5

4. Step 3. Design How to Start the Business Process

Create a Start Node in Your Business Process	4-1
Design the Communication Between a Client and the Business Process	4-2

5. Step 4. Create Task and Assign to User

Create Task and Assign to User Alternative 1	5-1
Create a Task Control.	5-2
Set the Default Values for the Task Control.	5-3
Define the Assignee and the Request Message	5-4
Specify the algorithm to Assign the Task	5-6
Create Task and Assign to User Alternative 2	5-9
Create a Task Control.	5-9
Create the Approve Resolution Task	5-10
Assign the Task to a User.	5-14
Specify a Due Date for Completion of the Task	5-16
Create Task and Assign to User Alternative 3	5-19
Create a Task Control.	5-19
Customize the Task Control.	5-20
Modify the Business Process	5-25

6. Step 5. Receive Resolution Approval From the Task Owner

Create an Event Choice Group to Handle the Resolution Approval Events.	6-2
Specify the Events That the Business Process Can Receive From the Task Control	6-4
Design How the Callback Events are Handled by the Resolution Approval Process. . . .	6-5

7. Step 6. Run the Resolution Approval Business Process

About This Document

This document explains how to build an application that can be used to orchestrate the integration of people in a business process. For an overview of the scenario used in this tutorial and an introduction to what you learn by completing the steps in the tutorial, see [“Tutorial: Building a Worklist Application” on page 1-1](#).

What You Need to Know

This document is intended for anyone who wants to learn how to integrate people in business processes using the Worklist system.

This document assumes a familiarity with building business processes in WebLogic Workshop. Before taking this tutorial, you should complete the following tutorials in the WebLogic Workshop online help system:

- [Tutorial: Building Your First Business Process](#)
- [Tutorial: Building Your First Data Transformation](#)

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the “e-docs” Product Documentation page at <http://e-docs.bea.com>.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File→Print option on your Web browser.

A PDF version of this document is available on the WebLogic Integration documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Integration documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at <http://www.adobe.com/>.

Related Information

The following WebLogic Integration documents contain information that is relevant to building Worklist solutions:

- [Using the Worklist System](#)
- [Worklist Administration](#) in [Managing WebLogic Integration Solutions](#)
- [Business Calendar Configuration](#) in [Managing WebLogic Integration Solutions](#)

Contact Us!

Your feedback on the BEA WebLogic Integration documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Integration documentation.

In your e-mail message, please indicate which version of the WebLogic Integration product and documentation you are using.

If you have any questions about this version of BEA WebLogic Integration, or if you have problems installing and running BEA WebLogic Integration, contact BEA Customer Support through BEA WebSupport at **www.bea.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates items that are displayed on the User Interface.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
<i>italics</i>	Indicates emphasis or book titles.
monospace text	<p>Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.</p> <p><i>Examples:</i></p> <pre>#include <iostream.h> void main () the pointer psz chmod u+w * \tux\data\ap .doc tux.doc BITMAP float</pre>
monospace boldface text	<p>Identifies significant words in code.</p> <p><i>Example:</i></p> <pre>void commit ()</pre>

Convention	Item
<i>monospace</i> <i>italic</i> <i>text</i>	Identifies variables in code. <i>Example:</i> String <i>expr</i>
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. <i>Examples:</i> LPT1 SIGNON OR
{ }	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
[]	Indicates optional items in a syntax line. The brackets themselves should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f <i>file-list</i>]... [-l <i>file-list</i>]...
	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
...	Indicates one of the following in a command line: <ul style="list-style-type: none"> • That an argument can be repeated several times in a command line • That the statement omits additional optional arguments • That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. <i>Example:</i> buildobjclient [-v] [-o name] [-f <i>file-list</i>]... [-l <i>file-list</i>]...
.	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Tutorial: Building a Worklist Application

WebLogic Integration's business process management (BPM) functionality enables the integration of diverse applications and human participants. The WebLogic Integration Worklist system enables people to collaborate in business processes—including assigning tasks, tracking the status of tasks, handling approvals, and so on.

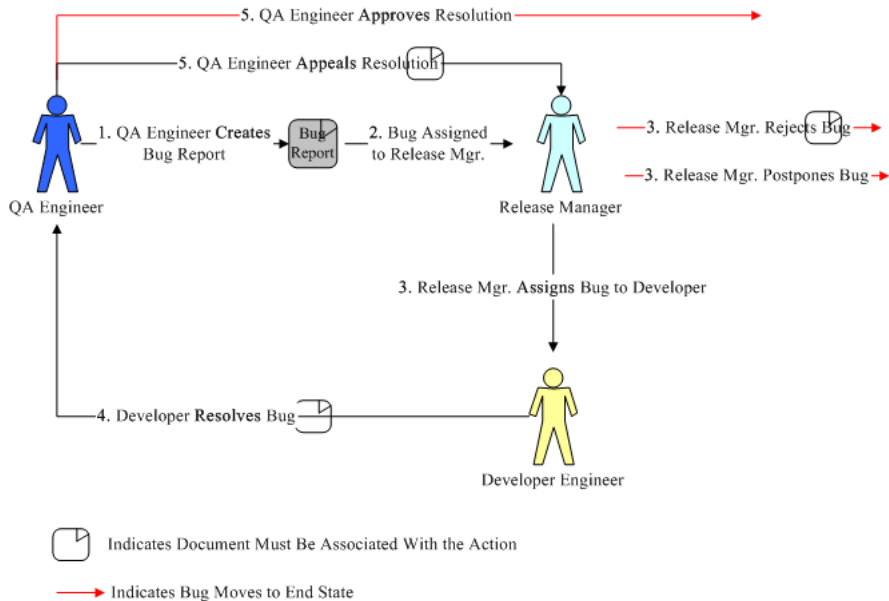
This tutorial provides a tour of the features available to design the interaction of people with business processes in the WebLogic Workshop graphical design environment. It describes how to create a business process that uses Worklist controls to orchestrate the resolution of a bug in a software company's bug tracking system.

Note: This document assumes a familiarity with building business processes in WebLogic Workshop. Before taking this tutorial, you should complete the following tutorials in the WebLogic Workshop online help system:

- [Tutorial: Building Your First Business Process](#)
- [Tutorial: Building Your First Data Transformation](#)

Tutorial Overview

The tutorial scenario is based on a portion of the life cycle of a software bug at a fictitious software company named SoftCo. The following figure and sequence of events outlines the path of a software bug through the SoftCo organization and the actors in the scenario:



1. A Quality Assurance engineer creates a bug report. To do so, they specify the following information in an online form: title of bug, description, a priority (1, 2, or 3), and the steps to reproduce the bug.
2. New bugs are assigned to a release manager with the fewest bugs in their queue. SoftCo policy states that the release manager must act on the bug report within two business days.
3. When a release manager receives a bug in their queue, they can reject it, pass it to the development engineers, or postpone to the next release:
 - **Reject**—If a release manager rejects a bug, they must provide documentation that records the reason for the rejection.
 - **Postpone**—Bugs that are postponed to a future release are reinstated to the release manager’s queue when the next release cycle begins.
 - **Assign**—When a bug is assigned to the development engineers group, any development engineer can review the bug and claim it if it is appropriate to do so.

4. Development engineers can resolve a bug as rejected or fixed:
 - **Reject**—Development engineers can reject the bug if it cannot be reproduced or if for some other reason it will not be fixed. They must provide documentation that records the reason for the rejection.
 - **Fix**—Development engineers can fix the bug. They must record the details of the fix in the bug report.
 5. When a bug is resolved as rejected or fixed, the Quality Assurance engineer who created the bug must either approve the resolution or appeal it. They must take action on the bug resolution within two business days.
 - **Appeal**—When a QA engineer appeals the resolution of a bug, they must provide documentation that describes the reason or reasons for the appeal. A bug that is appealed is routed back to the release manager who was originally assigned to review the bug. Bug resolutions that are not appealed or approved within two days are assumed approved.
 - **Approve**—When a QA engineer approves the resolution of a given bug, no further action is required for the bug.
 6. Release managers can reject appeals, in which case the bug becomes permanently resolved.
- Note:** To keep the team apprised of due dates, tasks that become overdue cause emails to be sent. If a bug is not claimed by a developer engineer, the release manager receives an email every two business days. If a bug is claimed by a developer engineer, but is not resolved within four business days, the developer engineer receives an email every two business days.

Values That Describe the Status of Software Bugs Include

NEW, NOT_REPRO, WILL_NOT_FIX, FIXED, POSTPONED, APPEALED, APPEAL_REJECTED

SoftCo Business Hours

SoftCo engineers work Monday through Friday 10AM to 8PM, and on Saturdays from 10AM to 2PM.

SoftCo Users and Groups

The following groups compose the team at SoftCo company: Quality Engineers, Release Managers, Development Engineers. Each worker is a member of one of these groups.

Steps in This Tutorial

The steps in this tutorial describe how to create a business process that uses Worklist controls to orchestrate the part of the bug tracking process that starts with the bug being resolved, as described in steps 4 and 5 of the life cycle of the preceding section.

In this scenario, a document that describes the resolution of the bug is created by the developer engineer—the business process you create by following the steps in this tutorial starts when it receives this document. Subsequently, the task of accepting or rejecting the resolution of the bug must be assigned to the QA engineer who created the bug in the first place. The QA engineer either approves or appeals the resolution of the bug, the business process sends a message to the client indicating whether the resolution is approved or appealed, and the business process terminates.

The tutorial includes the following steps:

Step 1. Set Up Your Environment

Learn how to set up the actors in the tutorial. That is, learn how to set up the users and groups required to complete the tasks you design in the Resolution Approval business process that you create. You also learn how to create a business calendar for your SoftCo enterprise.

Step 2. Create Your Application

Learn how to create a WebLogic Workshop application that holds the files you create as you work through this tutorial. Specifically, you create the starting application, and add the XML Schema files that you need when building the bug tracking application.

Step 3. Design How to Start the Business Process

Begin the design of the Resolution Approval business process. Specifically, you design how the business process is started at run time.

Step 4. Create Task and Assign to User

Learn how to build the part of the business process that orchestrates the assignment of the task of approving the resolution of a bug in the SoftCo bug tracking system. Learn to create and use Task controls in the business process.

Step 5. Receive Resolution Approval From the Task Owner

Learn how to design the business process to handle the possible events returned by the Task control.

Step 6. Run the Resolution Approval Business Process

Run and test the functionality of the business process you created. You use the WebLogic Workshop's browser-based interface to start the business process and the Worklist user

interface to play the role of a quality engineer assigned to the task of approving the resolution of the bug.

Tutorial: Building a Worklist Application

Step 1. Set Up Your Environment

In this step, you set up the actors in the tutorial, that is, the users and groups required to complete the tasks you design in the tutorial business process. Specifically, if you have not already done so, you create an integration domain for your worklist application. Next, you use the WebLogic Integration Administration Console to configure users and groups. Lastly, you create a business calendar for your SoftCo enterprise.

Complete the following tasks in Step 1:

- [Before You Begin](#)
- [Create a New WebLogic Integration Domain](#)
- [Configure the Users and Groups](#)
- [Create a Business Calendar](#)

Before You Begin

Before you begin the Worklist tutorial you must have WebLogic Platform 8.1 installed on your system. For more information, see [Installing WebLogic Platform](#) at the following URL:

<http://edocs.bea.com/platform/docs81/install/index.html>

Create a New WebLogic Integration Domain

The Worklist tutorial requires a WebLogic Integration domain. If you have not already created a WebLogic Integration domain, use the WebLogic Platform Configuration Wizard to create one. If you have already created an integration domain and you want to use it for the tutorial

Step 1. Set Up Your Environment

application, you can skip the procedure to create a new domain and instead start WebLogic Server in the existing domain.

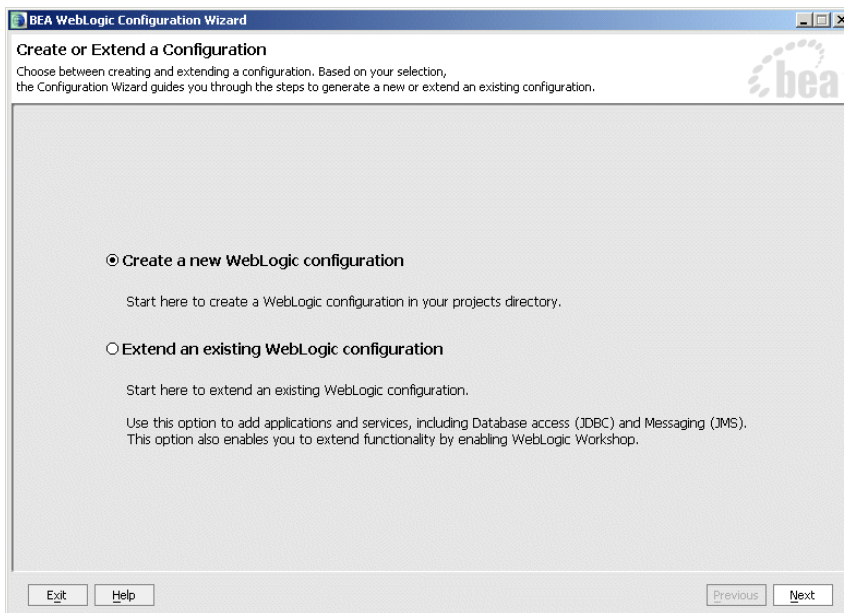
Note: The domain name used in this tutorial is `worktutorial`, but you can use any valid domain name.

To Create a New WebLogic Integration Domain

1. Start the Configuration Wizard:

- **Windows:** From the **Start** menu, choose **Programs→BEA WebLogic Platform 8.1→Configuration Wizard**.
- **UNIX:** Open a command shell, change to the `/common/bin` subdirectory of the product installation directory (such as `/bea/weblogic81/common/bin`), and enter the following command: `sh config.sh`.

The Configuration Wizard starts and displays the **Create or Extend a Configuration** screen.



2. Select **Create a new WebLogic configuration** (default), and click **Next**.
3. In the Configuration Wizard, make the following selections in each screen. After each selection, click **Next** to continue.

Table 2-1 Selections in the Configuration Wizard

Screen Name	Recommended Selection
Select a Configuration Template	Basic WebLogic Integration Domain (from the list of WebLogic Configuration Templates).
Choose Express or Custom Configuration	Express (default)
Configure Administrative Username and Password	Specify the administrator password (required) and change defaults as needed.
Configure Server Start Mode and Java SDK	Startup Mode: Development Mode (default) Java SDK Selection: Sun SDK (default)
Create WebLogic Configuration	In the Configuration Name field (lower-right corner), specify <code>worktutorial</code> or another name that you want to use for this domain. Go to the next step.

4. Click **Create**.

By default, the Configuration Wizard creates the new domain in the following location:

`BEA_HOME\user_projects\domains\domainName`, where:

- `BEA_HOME` is the directory in which you installed WebLogic platform, such as `c:\bea`.
- `domainName` is the name of the domain you created (`worktutorial`, in this case).

5. On the **Creating Configuration** screen, select **Start Admin Server**.

6. Click Done.

Configure the Users and Groups

You use the WebLogic Integration Administration Console to configure the following groups in your SoftCo enterprise: Quality Engineers, Release Managers, Development Engineers.

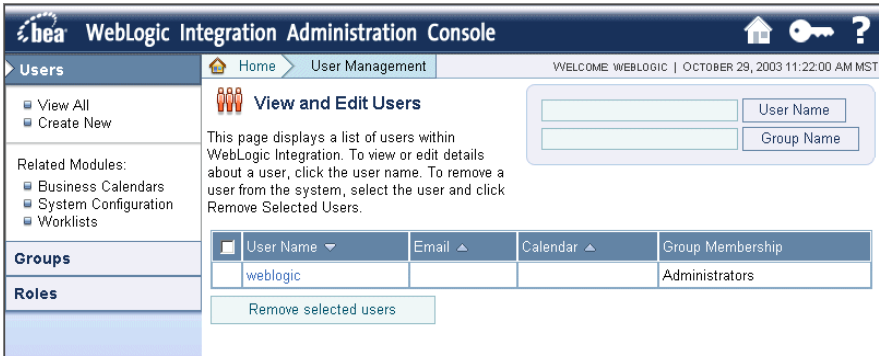
1. Start the WebLogic Integration Administration Console by doing either of the following:
 - From the Start menu, choose **Programs→BEA WebLogic Platform 8.1→Examples→WebLogic Integration→Integration Admin Console**.
 - In a Web browser, enter the following URL: `http://localhost:7001/wliconsole`.

Step 1. Set Up Your Environment

2. When prompted, enter the username and password that you specified for your domain.
The WebLogic Integration Administration Console displays the home page.
3. Select the **User Management** module:



The **User Management** screen is displayed:



4. In the left panel, click **Groups**. The **View and Edit Groups** screen is displayed, which allows you to create new groups.
5. In the left panel, click **Create New** to create a new group:
 - a. In the **Group Name** field, enter `QualityEngineers`.
 - b. In the **Group Membership** field, select **IntegrationUsers** from the list of **Available Groups** and use the arrow to move the **IntegrationUsers** group into the **Current Groups** list.
 - c. Click **Submit**.
6. Repeat Step 5 two more times to create two more groups—name the groups **ReleaseManagers**, and **DevelopmentEngineers**.
7. The groups you create are added to the list of groups on the **View and Edit Groups** screen:


<input type="checkbox"/>	Group Name ▲	Calendar ▼	Description	Group Membership
<input type="checkbox"/>	DevelopmentEngineers		SoftCo Development Engineers	IntegrationUsers
<input type="checkbox"/>	QualityEngineers		SoftCo QA Engineers	IntegrationUsers
<input type="checkbox"/>	ReleaseManagers		SoftCo Release Managers	IntegrationUsers

8. In the left panel, click **Users**. The **View and Edit Users** screen is displayed.
9. Click **Create New**. The **Add New User** screen is displayed.
10. Create users, as described in the following table:


User Name	Email	Password	Group Membership
QualityEngineerA	For simplicity, specify the same email address for each user, one that will work for your testing purposes.	For simplicity, specify the same password for each user.	QualityEngineers
			IntegrationUsers
QualityEngineerB			QualityEngineers
			IntegrationUsers
ReleaseManagerA			ReleaseManagers
			IntegrationUsers
ReleaseManagerB			ReleaseManagers
			IntegrationUsers
DevEngineerA			DevelopmentEngineers
			IntegrationUsers
DevEngineerB			DevelopmentEngineers
			IntegrationUsers

Your screen should resemble the following:

Step 1. Set Up Your Environment

 **View and Edit Users**





This page displays a list of users within WebLogic Integration. To view or edit details about a user, click the user name. To remove a user from the system, select the user and click Remove Selected Users.

	User Name ▾	Email ▲	Calendar ▲	Group Membership
<input type="checkbox"/>	DevEngineerA	worklistuser@softco.com		DevelopmentEngineers,IntegrationUsers
<input type="checkbox"/>	DevEngineerB	worklistuser@softco.com		DevelopmentEngineers,IntegrationUsers
<input type="checkbox"/>	QualityEngineerA	worklistuser@softco.com		IntegrationUsers,QualityEngineers
<input type="checkbox"/>	QualityEngineerB	worklistuser@softco.com		IntegrationUsers,QualityEngineers
<input type="checkbox"/>	ReleaseManagerA	worklistuser@softco.com		IntegrationUsers,ReleaseManagers
<input type="checkbox"/>	ReleaseManagerB	worklistuser@softco.com		IntegrationUsers,ReleaseManagers
	weblogic			Administrators

Create a Business Calendar

You must create a Business Calendar for use by the SoftCo enterprise during the tutorial scenario. A Business Calendar specifies the dates and times that an enterprise is *open for business*.

1. Navigate to the **Business Calendar Configuration** module in the WebLogic Integration Administration Console.

 **WebLogic Integration Administration Console**   


Business Calendars

- View All
- Create New
- Import Calendar


Related Modules:

- User Management

Business Calendar Mapping

 **Business Calendar Management**

This page displays a list of business calendars within WebLogic Integration. To view or edit details about a business calendar, click the calendar name. To export a business calendar, select the calendar and click Export.

	Calendar Name ▾	In Use ▲	Is System Calendar ▲
<input type="checkbox"/>	System Calendar	false	true

2. Click **Create New**. The **Create Business Calendar** screen is displayed.
3. In the **Business Calendar Name** field, enter `SoftwareTeamCalendar`.
4. Click **Create**. The calendar is created and listed on the **Business Calendar Management** screen.

5. In the calendar list, click **SoftwareTeamCalendar**. The **View Business Calendar Details** screen is displayed, which allows you to specify the time period rules.

The default rules in the business calendar specify that business hours for employees at SoftCo include Monday to Friday, 9AM to 5PM. You must change the default rules to create the appropriate calendar for the SoftCo enterprise. You can change the business rules for the calendar by editing the **Time Period Rules** table. To do so:

- a. For each of the days listed in the default calendar, click the Time Period to open an update page.
- b. Change the Start Hour and End Hour to **10** and **20**, respectively.

Recall that SoftCo engineers work Monday through Friday 10AM to 8PM and Saturdays from 10AM to 2PM.

- c. Click **Submit**. The business hours for Monday to Friday are updated in the **Time Period Rules** table.
- d. To add a rule for Saturday business hours, click **Add a New Rule**. A page is displayed in which you can specify the time period rules for Saturday. Select the following specifications in the appropriate fields:

Day of Week: Sat
Start Hour and Minute: 10 00
End Hour and Minute: 14 00
Free or Busy: Free

Click **Submit**. The business hours for Saturday are updated in the **Time Period Rules** table, as shown in the following figure:

Step 1. Set Up Your Environment

Business Calendar Name: SoftwareTeamCalendar
Time Zone: America/Los_Angeles
Is a system calendar false
[Edit Calendar Details.](#)

Time Period Rules:

	Time Periods	Free or Busy
<input type="checkbox"/>	Mon, 10:00AM - 8:00PM	Free
<input type="checkbox"/>	Tue, 10:00AM - 8:00PM	Free
<input type="checkbox"/>	Wed, 10:00AM - 8:00PM	Free
<input type="checkbox"/>	Thu, 10:00AM - 8:00PM	Free
<input type="checkbox"/>	Fri, 10:00AM - 8:00PM	Free
<input type="checkbox"/>	Sat, 10:00AM - 2:00PM	Free

This completes creating the Business Calendar for the tutorial. To begin creating a WebLogic Workshop application that orchestrates the resolution of a bug in the SoftCo bug tracking system, proceed to [Chapter 3, “Step 2. Create Your Application.”](#)

Note: To learn more about Business Calendars and defining the time-period rules, see [Business Calendar Configuration](#) at the following URL:

<http://edocs.bea.com/wli/docs81/manage/businesscalendar.html>

Step 2. Create Your Application

In this step, you create a WebLogic Workshop application. This application holds the files you create as you work through this tutorial. Specifically, you create the starting application and add the XML Schema files that you need to build the bug tracking application. In subsequent steps, you build the business process that orchestrates the resolution of a bug in the SoftCo bug tracking system. The tasks in this step include:

- [Start WebLogic Workshop](#)
- [Create a WebLogic Workshop Application](#)
- [Create the Tutorial Schemas](#)

Start WebLogic Workshop

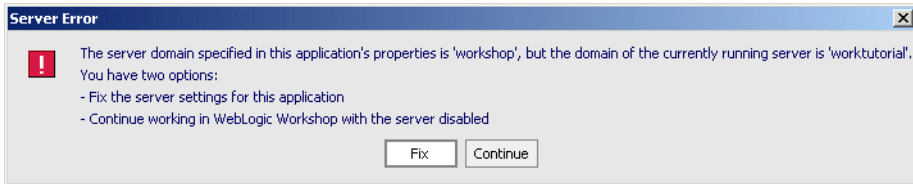
Complete the following steps to start WebLogic Workshop and select the integration server:

1. Start WebLogic Workshop, as described in [How Do I: Start WebLogic Workshop?](#) in the WebLogic Workshop Help, which is available at the following URL:

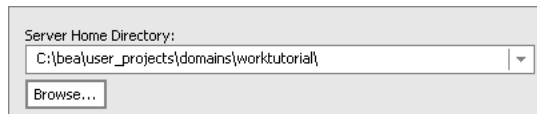
`http://edocs.bea.com/workshop/docs81/doc/en/integration/howdoI/howStartWorkshop.html`

2. If a server error dialog box appears, click **Fix**; otherwise, go to [“Create a WebLogic Workshop Application” on page 3-2](#).

Step 2. Create Your Application



3. After the **Application Properties** screen appears, click **Browse** beneath the **Server Home Directory** list box.
4. In the **Select a WebLogic Server config.xml** window, select the following:
`BEA_HOME\userprojects\domains\worktutorial\config.xml`
5. Click **Open**. The **Application Properties** screen shows the following:

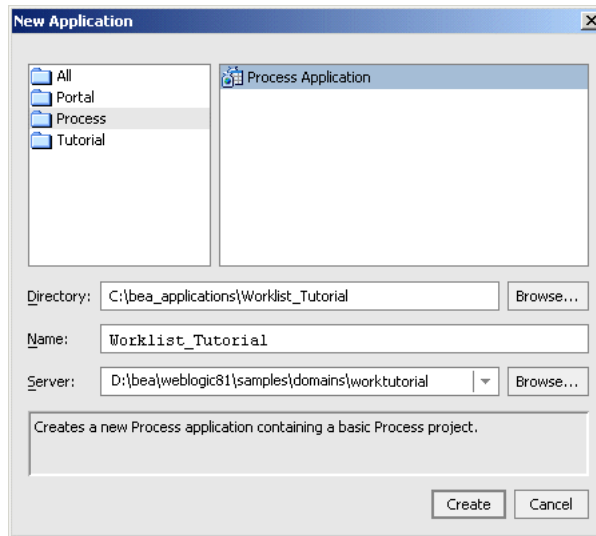


6. Click **OK**.

Create a WebLogic Workshop Application

Complete the following steps to create a WebLogic Workshop application in which to build the Bug Resolution business process:

1. Close any open applications. From the WebLogic Workshop menu, click **File→Close Application**.
2. From the WebLogic Workshop menu, click **File→New→Application**. The **New Application** dialog box is displayed.



3. In the **New Application** dialog box, select **Process** in the left pane, and select **Process Application** in the right pane.
4. In the **Directory** field, select the directory in which you want to create your application. The paths used in the remainder of this tutorial assumes that the application is created in `C:\bea_applications\`. If you create the directory in a different location or use another name, adjust the instructions accordingly.
5. In the **Name** field, enter `Worklist_Tutorial`.

6. Click the arrow in the **Server** field to display a drop-down list of servers. Then choose an integration domain. For example, if you created a new integration domain in the default location, the path to the integration server is:

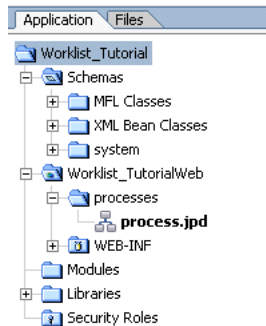
`BEA_HOME\user_projects\domains\worktutorial`

where `BEA_HOME` is the directory in which you installed WebLogic Platform.

7. Click **Create**.
8. Your Worklist Tutorial Application is created and displayed in the **Application** panel.

Note: If the **Application** panel is not visible in WebLogic Workshop, choose **View→Application** from the menu bar.

Step 2. Create Your Application



The **Application** panel displays a hierarchical representation of the files and resources available in your application. The components in this tutorial include the following:

Worklist_Tutorial—The application folder.

Schemas—A Schemas project that contains the system XML Schemas used in the application.

Worklist_TutorialWeb—A Web application project folder. Every application contains one or more projects. Projects represent WebLogic Server Web applications. In other words, when you create a project, you are creating a Web application. (The name of your project is included in the URL your clients use to access your application.)

Web Applications are J2EE deployment units that define a collection of Web resources such as business processes, Web services, JSPs, servlets, HTML pages, and can define references to external resources such as EJBs.

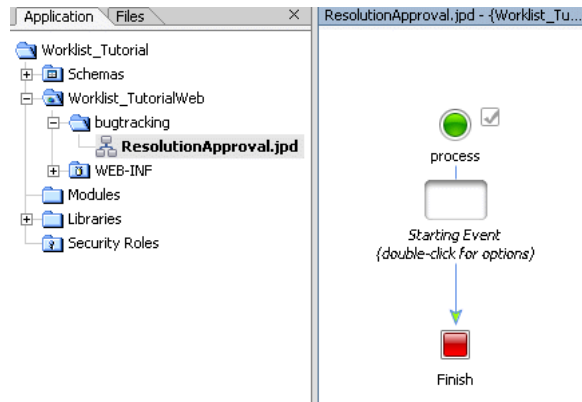
Note: The Web application project folder is named by appending **Web** to the name you gave your application.

processes—Located in the Worklist_TutorialWeb folder. This is the default folder created when you create a new application—your project files are located in this folder. By default, one file is created in a new Process application: **process.jspd**.

9. Rename the **processes** folder and the **process.jspd** file.
 - a. In the **Application** tab, click the + beside the **Worklist_TutorialWeb** folder to view the **processes** folder.
 - b. Right-click the **processes** folder and select **Rename** from the drop-down menu.
 - c. Enter **bugtracking** as the new folder name, and press **Enter** on your keyboard.

- d. Right-click **process.jspd** in the **Application** tab and select **Rename** from the drop-down menu.
- e. Enter `ResolutionApproval.jspd` as the new name, and press **Enter** on your keyboard.
- f. Double-click **ResolutionApproval.jspd**.

The **Application** tab and the **Design View** now resemble the following figure:



Create the Tutorial Schemas

In this tutorial, XML schemas are used to validate the XML data that is exchanged between the Worklist and the Bug Resolution business process. This section describes how to create the schemas in the **Schemas** project in your Worklist Tutorial application.

To Create the Tutorial Schemas

1. In the **Application** tab, right-click the **Schemas** folder.
2. From the drop-down menu, select **New→XML Schema**. The **New File** dialog box is displayed.
3. Enter `BugResolution.xsd` as the name of your schema file, and click **Create**. The **BugResolution.xsd** file is created and displayed in the **Application** panel.

When you create an XSD, a build of the **Schemas** project folder is triggered. The build verifies that the schema file is well formed. It also verifies that the element and attribute names in the XML Schema do not conflict with the XSD files that are in the **Schemas** project. For more information about what gets generated when you import schemas, see [Importing Schemas](#), which is available at the following URL:

Step 2. Create Your Application

<http://edocs.bea.com/workshop/docs81/doc/en/integration/dtguide/dtguidemapperimportschemas.html>

4. Copy the contents of [Listing 3-1](#) and replace the default schema file in the **Source View**.

Listing 3-1 BugResolution.xsd

```
<?xml version="1.0"?>
<xs:schema elementFormDefault="qualified"
attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.bea.com/WLIWorklistTutorial/BugResolution.xsd"
  targetNamespace="http://www.bea.com/WLIWorklistTutorial/BugResolution.
xsd">

  <xs:element name="bug-resolution" type="tns:bug-resolution-type"/>

  <xs:complexType name="bug-resolution-type">
    <xs:sequence>
      <xs:element name="description" type="xs:string"/>
      <xs:element name="resolution-code" type="xs:string"/>
      <xs:element name="resolution-text" type="xs:string"/>
      <xs:element name="bug-creator" type="xs:integer"/>
    </xs:sequence>
  </xs:complexType>

</xs:schema>
```

5. Repeat steps 1 through 4, but this time create an XML schema file named **ResolutionAppeal.xsd**. Copy the contents of [Listing 3-2](#) and paste it into the **Source View** to create a the **ResolutionAppeal.xsd** schema file in your **Schemas** folder.

Listing 3-2 ResolutionAppeal.xsd

```
<?xml version="1.0"?>

<xs:schema elementFormDefault="qualified"
attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tns="http://www.bea.com/WLIWorklistTutorial/ResolutionAppeal.xsd"
  "
```

```

    targetNamespace="http://www.bea.com/WLIWorklistTutorial/ResolutionAppe
al.xsd">

    <xs:element name="resolution-appeal"
type="tns:resolution-appeal-type"/>

        <xs:complexType name="resolution-appeal-type">
<xs:sequence>
    <xs:element name="appeal-text" type="xs:string"/>
    <xs:element name="resolution-code" type="xs:string"/>
    <xs:element name="resolution-text" type="xs:string"/>
</xs:sequence>
</xs:complexType>

</xs:schema>

```

6. Select **File**→**Save All** to save your work.

This completes the set up of your new application. To begin the work of adding the business logic to the Resolution Approval business process (ResolutionApproval.jpd), proceed to [Chapter 4, “Step 3. Design How to Start the Business Process.”](#)

Note: Recall from Step 3 that when you create an XSD, a build of the Schemas project folder is triggered.

Step 2. Create Your Application

Step 3. Design How to Start the Business Process

In this step, you begin the design of the Resolution Approval business process. Specifically, you design how the business process is started at run time.

To start this step, you design the **Start** node in your business process to receive a *Bug Resolution* message from a client—the receipt of this message is the trigger that starts the business process. You then create a variable to hold the incoming message.

In **Design View**, interactions between a business process and a client application are represented by **Client Request** and **Client Response** nodes. In this tutorial, you add a **Client Request** node to your business process and subsequently create the code on this node to handle the receipt of a message from a client.

Complete the following tasks to design the **Client Request** node that starts your business process:

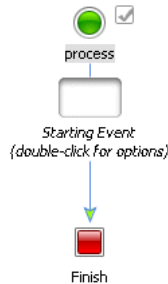
- [Create a Start Node in Your Business Process](#)
- [Design the Communication Between a Client and the Business Process](#)

Create a Start Node in Your Business Process

In [Step 2. Create Your Application](#), you created a business process in the Worklist Tutorial application. This step begins the design of that business process with the bug tracking business logic.

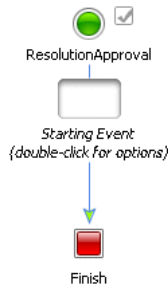
1. With the `Worklist_Tutorial` open in WebLogic Workshop, double-click **ResolutionApproval.jspd** on the **Application** panel. The business process is displayed in **Design View**.

Step 3. Design How to Start the Business Process



2. Rename the business process to better represent its function:
 - a. Click the name (**process**) in the **Design View**.
 - b. Press **F2** and enter **ResolutionApproval** as the name of the node, then press **Enter** on your keyboard.

The business process in **Design View** is named **ResolutionApproval** to match the name you gave the JPD file in the **Application** tab.



Design the Communication Between a Client and the Business Process

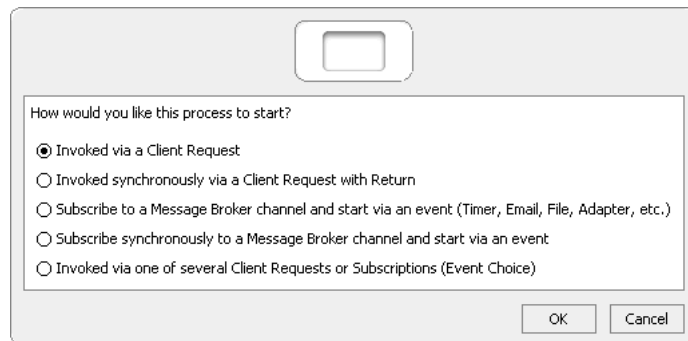
In this step, you set up the communication between a client and the business process. To do so, complete the steps in the following sections:

- [To Design Your Client Request Node](#)
- [To Specify General Settings](#)
- [To Specify Receive Data](#)

To Design Your Client Request Node

Designing your **Client Request** node includes creating a method and parameters that your client uses to trigger the start of your business process, plus designing the logic for handling the receipt of a request from a client.

1. In **Design View**, double-click the empty **Starting Event** target to display the Start node builder. The node builder shows the following:



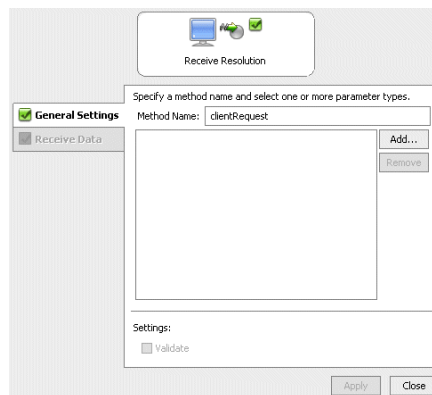
Note: Node builders provide a task-driven user interface that helps you design your the communication between a business process and its clients and other resources.

2. Select **Invoked via a Client Request**.
3. Click **OK**. The node builder closes and the empty node that was associated with the **Start** node is now populated with a **Client Request** node.
4. Rename the **Client Request** node.
 - a. Click the **Client Request** node, then press **F2**.
 - b. Enter `Receive Resolution` to replace the name `Client Request` for the node.
 - c. Press **Enter** on your keyboard. Your business process should now resemble the following figure:

Step 3. Design How to Start the Business Process



5. Double-click the **Receive Resolution** node to invoke the node builder, as shown in the following figure:



To Specify General Settings

The following steps describe how to specify the method exposed by your business process to clients—clients invoke this method to start and make requests on your business process.

1. In the **Method Name** field on the **General Settings** tab, change the default method name from `clientRequest` to `receiveResolution`.

Note: When you make your business process available as a service, the name you assign to a method on a **Client Request** node is the name of the method that is exposed via the Web Services Description Language (WSDL). In general, we recommend that you define a name that is representative of the service offered by your business process.

2. Specify a data type for the parameter to your `receiveResolution` method:

- a. On the **General Settings** tab, click **Add**. A panel, which shows the data types you can use is displayed:

☒ XML ☐ NonXML ☐ Java

The Bug Resolution message from clients is an XML message. Therefore, we are concerned with **XML Types** at this node.

- b. If necessary, select **XML**. The panel is populated with a list of XML Schema files (*Typed XML*) and a list of *Untyped* XML objects available in your project.

Note: Remember that you added the required XML Schemas as you built the Receive Resolution Business process in [“Create the Tutorial Schemas” on page 3-5](#).

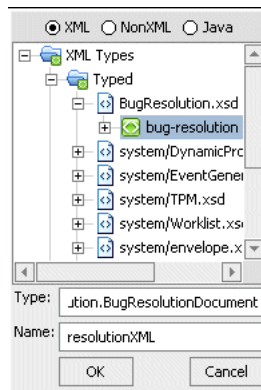
In this step, you will use the `BugResolution.xsd` to specify the structure of documents that clients can send to start your business process.

- c. In the list of **XML Types**, click the + associated with `BugResolution.xsd`.

A graphical representation of the XML Schema defined by `BugResolution.xsd` is displayed in the data types panel.

- d. Click the **bug-resolution** node. (It represents the parent element in your XML document.) The **Type** field is populated with the XML type:

`com.bea.wliWorklistTutorial.bugResolution.BugResolutionDocument`.



- e. In the **Name** field, replace the default parameter name (**x0**) with **resolutionXML**.
- f. Click **OK**. The parameter specifications you made is displayed in **General Settings** tab in the node builder; the parameter type is **BugResolutionDocument** and the parameter name is **resolutionXML**.

Step 3. Design How to Start the Business Process

You have now specified the method exposed to clients by your business process. Messages from clients are expected to be *typed* XML. This means that the messages received from clients must contain XML that is valid against an XML Schema (in this case, `BugResolution.xsd`).

To Specify Receive Data

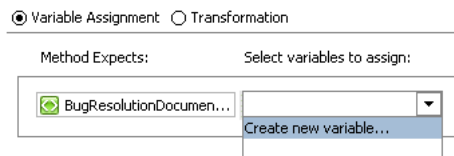
1. Click the **Receive Data** tab. This tab allows you to specify a variable to which a Bug Resolution message received from a client is assigned at run time. By default, the **Receive Data** tab opens on the **Variable Assignment** panel.

Receive Data tabs have two modes:

- **Variable Assignment**—Use this mode when you want to assign the data received from the client to a variable of the same data type.
- **Transformation**—Use this mode when you want to create a transformation between the data assigned to a variable and that expected by the method parameter.

In this tutorial, we use the **Variable Assignment** mode because we want to assign the XML message received from the client directly to a variable of the same data type. Specifically, you create a variable of typed XML (`BugResolutionDocument`) to which your business process assigns the incoming Bug Resolution from clients.

2. Under **Select variables to assign**, click the arrow in the drop-down list and select **Create new variable...**



3. In the **Create Variable** dialog box, take the following steps:
 - a. In the **Variable Name** field, enter `resolutionXML`.
 - b. In the **Select Variable Type** field, make sure that the **bug-resolution** element is selected (in `BugResolution.xsd` in the list of **XML Types**)
 - c. Click **OK**. Your new variable is created and displayed in the **Receive Data** tab. Note that in the **Data Palette**, the **resolutionXML** variable is also listed as an **XML** variable.
4. Click **Apply**.

Both tabs in the node builder (**General Settings** and **Receive Data**) are marked complete



5. Click **Close**. The **Client Receive** node builder closes.

This step completes the design of the Start node for your business process.

You have now completed the design of how the business process is started at run time. To begin integrating your business process with a user using a Worklist control, proceed to [Chapter 5, “Step 4. Create Task and Assign to User.”](#)

Step 3. Design How to Start the Business Process

Step 4. Create Task and Assign to User

This step describes how to design a common pattern in worklist business processes—one that creates a task, assigns the task to a user, and specifies a date by which the completion of the task is due. You design this integration of your business process with a user using a Worklist control, specifically the Task control.

This section provides two methods for creating a task and assigning the task to a user:

- [Create Task and Assign to User Alternative 1](#)—This alternative is the preferred method. It is more intuitive than Alternative 2 and compacts four operations into one.
- [Create Task and Assign to User Alternative 2](#)—This alternative shows you a less efficient way to design a business process. When compared to Alternative 1, it can help avoid common pitfalls and more effectively use the features of Worklist.
- [Create Task and Assign to User Alternative 3](#)—This alternative shows the power of the Worklist and shows you how to build your own Task control that is customized to your needs.

Note: BEA recommends that you create a new business process (JPD) for each alternative.

Create Task and Assign to User Alternative 1

This alternative is the preferred method. It is more intuitive than Alternative 2 and compacts four operations into one.

This section includes the following tasks:

- [Create a Task Control](#)

Step 4. Create Task and Assign to User

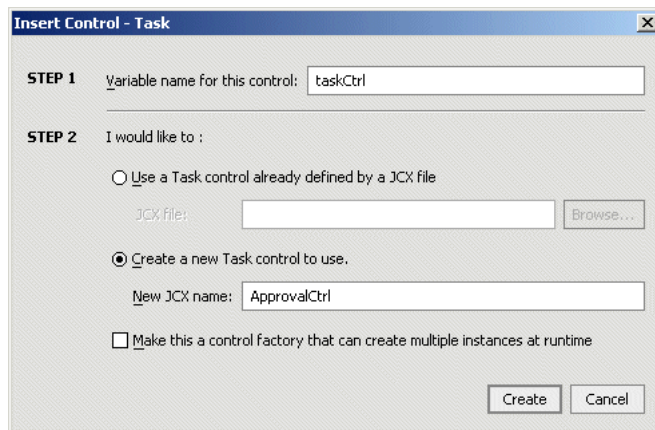
- Set the Default Values for the Task Control
- Define the Assignee and the Request Message
- Specify the algorithm to Assign the Task

Create a Task Control

A Task control creates and manages an instance of a Task. To create the Task control that manages the task of approving the resolution of a bug in our SoftCo enterprise, complete the following:

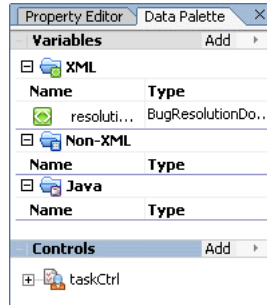
1. If the business process is not displayed in the **Design View** tab, in the **Application** panel, double-click **ResolutionApproval.jpdl**.
2. On the **Data Palette**, in the **Controls** tab, click **Add→Integration Controls→Task**. The **Insert Control - Task** dialog box is displayed.

Note: If the Controls tab is not visible in WebLogic Workshop, click **View→Windows→Data Palette** from the menu bar.



3. In **Step 1**, enter `taskCtrl` as the name for the instance of this control.
4. In **Step 2**, select **Create a new Task control to use**. Then, in the **New JCX name** field, enter `ApprovalCtrl`.
5. Click **Create**. A new Task control and an instance of it are created and the **Insert Control** dialog box is closed.

A new JCX file, named `ApprovalCtrl.jcx`, is created and displayed on the **Application** tab. The instance of the control is displayed on the **Controls** tab of the **Data Palette**.



Set the Default Values for the Task Control

A Task control lets you specify some of the control's default values whenever a task is created. These values can be overridden by the arguments in the creation method.

1. On the **Data Palette**, click **taskCtrl**.
2. Click the **Property Editor** tab.

The **Property Editor** displays a list of properties for the Worklist control. Notice that some values are already set. These values can be modified.

3. On the **Property Editor**, in the **Task** group, click the **name** property, then enter `approveResolutionTask`.
4. Click the **completion-due-business-date** property, then click . The **Property Text Editor** is displayed.
5. In the **Property Text Editor**, enter `2d`.

Note: If you do not specify a valid business date, the syntactic analyzer catches the error when looking at the control declaration and displays it in the **Source View** of the JPD.

The default control values are now defined for creating a task, as shown in the following figure:

Step 4. Create Task and Assign to User

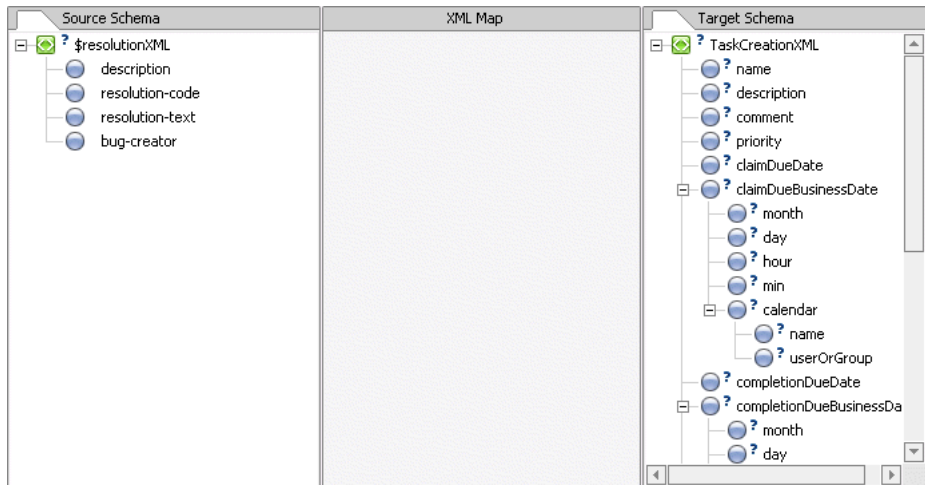
approveResolutionTask - Control	
general	
name	taskCtrl
security	
callback-roles-allowed	
task	
name	approveResolutionTask
description	
comment	
priority	1
owner	
assignee	
algorithm	ToUsersAndGroups
user	
group	
advanced	
can-be-reassigned	true
can-be-returned	true
can-be-aborted	true
claim-due-business-date	
completion-due-business-c	2d
completion-user-calendar	
completion-calendar	system
claim-user-calendar	
claim-calendar	system

Define the Assignee and the Request Message

Define the assigned and request message using a single XML operation and the XML Mapper, as follows:

1. Click the **Data Palette** tab.
2. If the available methods on the **taskCtrl** control are not visible, click the + beside the control to expand the list.
3. Select the following method: `String createTask(TaskCreationXMLDocument arg0)`. Then drag and drop it onto the business process in **Design View**, placing it on the process immediately after the **Receive Resolution** node.
4. Double-click the **createTask** node. The node builder opens with the **General Settings** tab displayed.
5. Open the **Send Data** tab. By default, the **Send Data** tab opens with the **Variable Assignment** pane displayed.
6. Select **Transformation**.

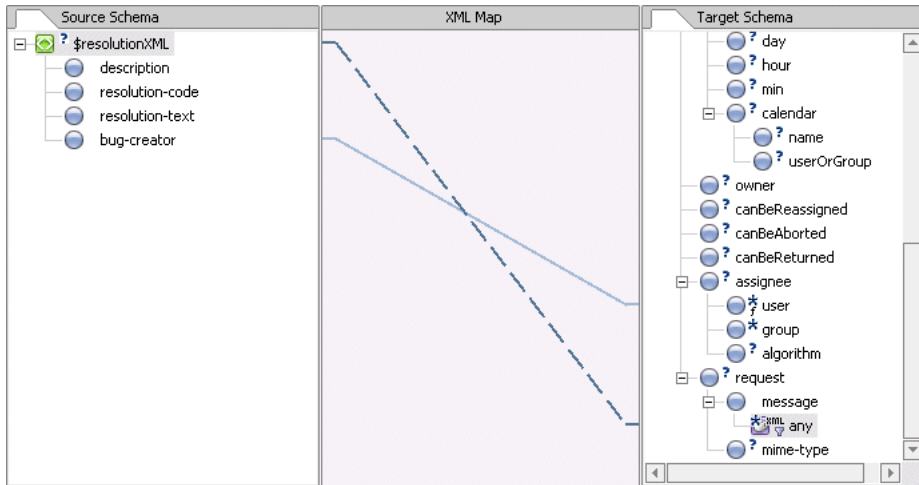
7. In **Step 1**, click **Select Variable**, then click **ResolutionXML (BugResolutionDocument)**.
8. In **Step 2**, click **Create Transformation**. The Transformation tool opens, which displays a representation of the BugResolution XML document in the **Source Schema** pane, and a **String** in the **Target Schema** pane.



This figure shows the XML mapper in the center column. The left column shows the schema of the XML that is expected by the create method. The right column shows the schema of the XML that is received. All the elements in this schema are optional. The only thing we need to be sure of when creating a task is that it has a name. You do not need to worry about setting a name in this tutorial because you have already set default name value for this control: **taskCtrl**.

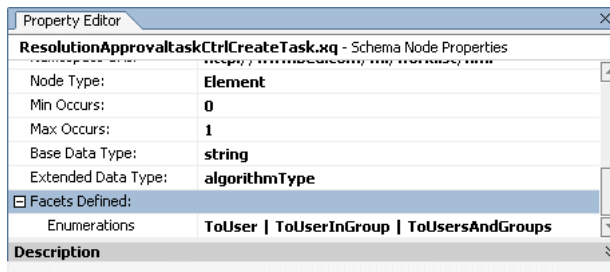
9. In the **Source Schema** pane, click **bug-creator** and drag your mouse pointer over to **assignee-user** in the **Target Schema** pane. A line is drawn between the **bug-creator** and the **assignee-user** elements in the **XML Map** pane.
10. In the **Source Schema** pane, click **\$resolutionXML** and drag your mouse pointer over to **request-message-any** in the **Target Schema** pane. A line is drawn between the **\$resolutionXML** and **request-message-any** elements in the **XML Map** pane.

Step 4. Create Task and Assign to User



Specify the algorithm to Assign the Task

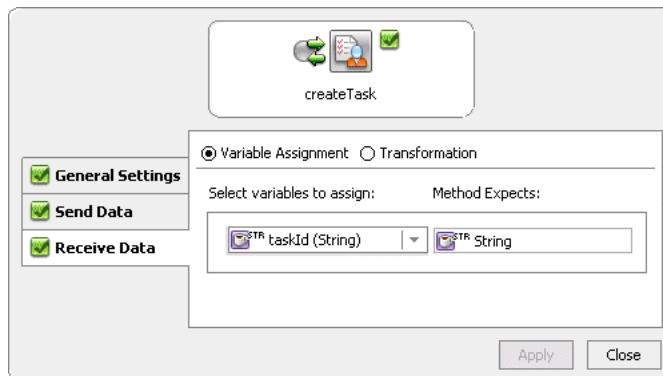
By default, the algorithm `AssignToUsersAndGroups` is used. However, in this tutorial we want the task to be assigned to a specific user and to be claimed by this user. To see a list of the accepted values, in the **Target Schema** pane, click the **assignee-algorithm** element. The **Facets Defined** property in the **Property Editor** provides the list of accepted values.



To Specify Which Algorithm is Used to Assign the Task

1. Right-click the **assignee-algorithm** element, select **Create Constant** from the menu.
2. In the **Create Constant** dialog box, enter `ToUser`, and then click **OK**.
3. In the **Application** pane, double-click **ResolutionApproval.jpdl** to return to the node builder.
4. If the node builder is not displayed in **Design View**, click the **Design View** tab.

5. Open the **Receive Data** tab.
6. By default, the **Receive Data** tab opens on the **Variable Assignment** pane. The **Method Expects** field is populated with the data type expected to be returned by the `createTask()` method: `String`.
7. Under **Select variables to assign**, click the arrow in the drop-down list and select **Create new variable**. The **Create Variable** dialog box is displayed with Java selected as the variable type.
8. In the **Create Variable** dialog box, in the **Variable Name** field, enter `taskId`.
9. Click **OK**. Your new variable (of type **String**) is created and displayed in the **Receive Data** tab.
10. Click **Apply**. All tabs in the node builder should be marked complete.



11. Click **Close** to save the specifications you made to the **assignTaskByName** node and close its node builder.

The Resolution Approval business process should now resemble the following figure:

Step 4. Create Task and Assign to User



This completes the design of a business process that creates a task, assigns the task to a user, and specifies a date by which the completion of the task is due using a Worklist control. This step also assigned the task of approving the resolution of a bug to the user who created the bug.

This method is more intuitive than the [Create Task and Assign to User Alternative 2](#) and compacts four operations into one. However, it is worthwhile to examine the [Create Task and Assign to User Alternative 2](#) to deepen your understanding of the Worklist and utilize its functionality more fully.

To design your business process to handle the other possible events for the resolution of the bug, proceed to [Chapter 6, “Step 5. Receive Resolution Approval From the Task Owner.”](#)

Create Task and Assign to User Alternative 2

This alternative shows you a less efficient way to design a business process. When compared to Alternative 1, it can help avoid common pitfalls and more effectively use the features of Worklist.

This section includes the following tasks:

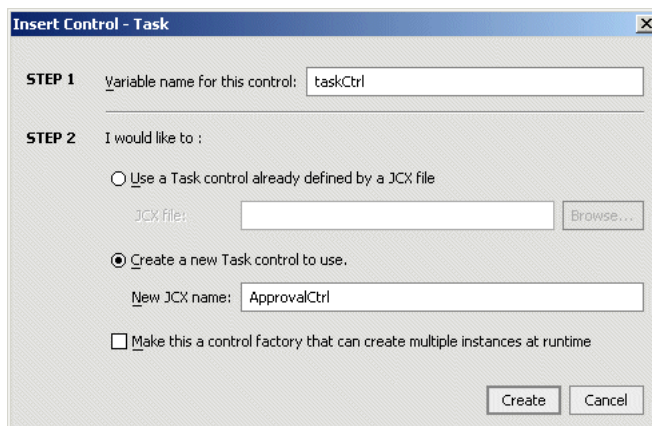
- [Create a Task Control](#)
- [Create the Approve Resolution Task](#)
- [Assign the Task to a User](#)
- [Specify a Due Date for Completion of the Task](#)

Create a Task Control

A Task control creates and manages an instance of a Task. To create the Task control that manages the task of approving the resolution of a bug in our SoftCo enterprise, complete the following:

1. If the business process is not displayed in the **Design View** tab, in the **Application** panel, double-click **ResolutionApproval.jpdl**.
2. On the **Data Palette**, in the **Controls** tab, click **Add→Integration Controls→Task**. The **Insert Control - Task** dialog box is displayed.

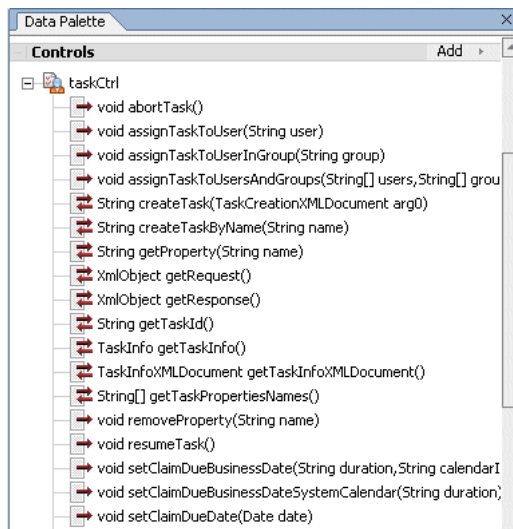
Note: If the Controls tab is not visible in WebLogic Workshop, click **View→Windows→Data Palette** from the menu bar.



Step 4. Create Task and Assign to User

3. In **Step 1**, enter `taskCtrl` as the name for the instance of this control.
4. In **Step 2**, select **Create a new Task control to use**. Then, in the **New JCX name** field, enter `ApprovalCtrl`.
5. Click **Create**. A new Task control and an instance of it are created and the **Insert Control** dialog box is closed.

A new JCX file, named `ApprovalCtrl.jcx`, is created and displayed on the **Application** tab. The instance of the control is displayed on the **Controls** tab of the **Data Palette**. On the **Data Palette**, expand the control instance by clicking the + beside its name to display the base methods provided for this control.



Note: For a complete list of the base methods on Task controls, see [Using Worklist Controls](http://edocs.bea.com/wli/docs81/worklist/index.html) in *Using the Worklist*, which is available at the following URL:

<http://edocs.bea.com/wli/docs81/worklist/index.html>

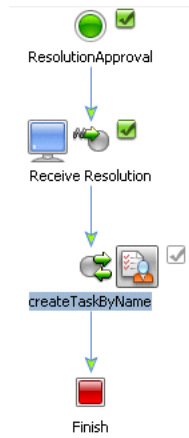
Create the Approve Resolution Task

In this step, you design the interaction of the business process with the Task control in **Design View** by simply dragging and dropping the Task control methods from the **Data Palette** onto the point in your business process at which you want to create the appropriate logic. In this tutorial, you create one node in the process that creates the task and another node that sends the data about the task to the Task control. To do so, complete the steps in the following sections:

- To Create the Approve Resolution Task
- To Send Information About the Task to the Task Control

To Create the Approve Resolution Task

1. If necessary, expand the its list of available methods by clicking the + beside the **taskCtrl** control on the **Data Palette**.
2. Select the following method: `String createTaskByName(String name)`. Then drag and drop it onto the business process in **Design View**, placing it on the process immediately after the **Receive Resolution** node.



A **Control Send with Return** node is created that represents the synchronous call to your **taskCtrl** control. The node is named according to the name of the method you dragged onto the business process. In this case: **createTaskByName**.

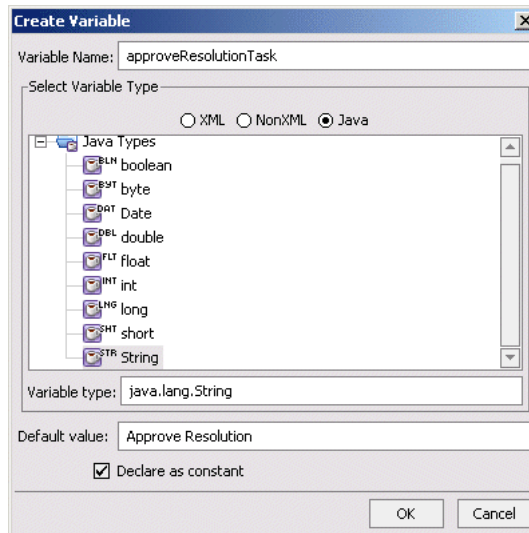
Note: This interaction is designed to be synchronous, meaning that the business process blocks waiting for a response from the control. In the case of the `createTaskByName` method, the response is a `String` that contains the Task ID. A unique Task ID is assigned to each task created using an instance of the Task control.

3. Double-click the **createTaskByName** node. The node builder opens with the **General Settings** tab displayed. The Control instance and target methods are already selected: **taskCtrl** and **String createTaskByName(String name)**, respectively.
4. Open the **Send Data** tab.

Step 4. Create Task and Assign to User


By default, the **Send Data** tab opens with the **Variable Assignment** pane displayed. The **Method Expects** field is populated with the data type expected by the `createTaskByName()` method exposed by the **taskCtrl** control: `String name`.

5. Under **Select variables to assign**, click the arrow in the drop-down list and select **Create new variable...** The **Create Variable** dialog box is displayed with Java selected as the variable type.



6. In the **Create Variable** dialog box, complete the following steps:
 - a. In the **Variable Name** field, enter `approveResolutionTask`.
 - b. The **Variable type** field should contain `java.lang.String`.
 - c. In the **Default Value** field, enter `Approve Resolution`. In this way, you specify the name of the task.
 - d. Select **Declare as constant**.
 - e. Click **OK**. Your new variable (of type `String`) is created and displayed in the **Receive Data** tab. Note that the **approveResolutionTask** variable is also listed as a Java variable in the **Data Palette**.
7. Open the **Receive Data** tab.

By default, the **Receive Data** tab opens on the **Variable Assignment** pane. The **Method Expects** field is populated with the data type expected to be returned by the `createTaskByName()` method: `String`.

8. Under **Select variables to assign**, click the arrow in the drop-down list and select **Create new variable...** The **Create Variable** dialog box is displayed with Java selected as the variable type.
9. In the **Create Variable** dialog box, complete the following steps:
 - a. In the **Variable Name** field, enter `taskId`.
 - b. Click **OK**. Your new variable (of type `String`) is created and displayed in the **Receive Data** tab.
10. Click **Apply**. All tabs in the node builder should be marked complete .
11. Click **Close**. The specifications you made to the **assignTaskByName** node are saved and the node builder closes.

To Send Information About the Task to the Task Control

In this tutorial, you designed the business process to be invoked when it receives a Bug Resolution document from a client. The data contained in the Bug Resolution document can be used by the Task control when tasks, due dates, and so on are being assigned. In this step, you create a node in the business process to send the Bug Resolution document data to the Task control. To do so, complete the following steps:

1. If necessary, expand the list of available methods on the **Data Palette** by clicking the + beside the **taskCtrl** control.
2. Select the following method: `void setRequest(XmlObject xml)`. Then in **Design View** drag and drop it onto the business process, placing it on the business process immediately after the **createtaskByName** node. A new node (**setRequest**) is created.
3. Double-click the **setRequest** node.
4. Open the **Send Data** tab. Its node builder opens with the **Variable Assignment** panel selected by default.
5. Under **Select variables to assign**, click the arrow and select **resolutionXML (BugResolutionDocument)**.
6. Click **Apply**, then **Close**. The specifications you made on this node are saved and its node builder is closed.

Step 4. Create Task and Assign to User

The Resolution Approval business process should now resemble the following figure:



Assign the Task to a User

In this step you assign the `approveResolution` task you created in the preceding step to a user in the SoftCo enterprise. You do so in the same way as you created the task—that is, in the **Design View**, you drag and drop a Task control method from the **Data Palette** onto the point in your business process at which you want to design the interaction.

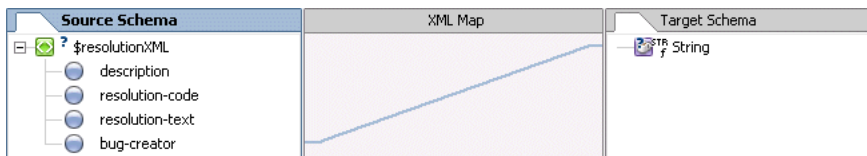
1. On the **Data Palette**, from the **taskCtrl**, select the following method: `void assignTaskToUser(String user)`. Then in **Design View**, drag and drop it onto the business process, placing it on the process immediately after the **setRequest** node.
2. Double-click the **assignTaskToUser** node in the **Design View**, then open the **Send Data** tab to specify a variable from which you can get the user name to assign. The **Send Data** tab has two modes:
 - **Variable Assignment**—Use this mode when you want to directly assign a variable that holds the data expected by the method.
 - **Transformation**—Use this mode when you want to create a transformation between the data assigned to a variable and that expected by the method parameter.

For this part of the tutorial, you must switch to the **Transformation** mode because the data type required as input to the **taskCtrl** control is a Java `String` type and the variable in

which the Bug Resolution message is stored is of type XML, that is, the `BugResolutionDocument`. Note that the Bug Resolution message includes the names of the users to which a bug can be assigned and the `BugResolutionDocument` is valid against the `BugResolution.xsd` schema.

WebLogic Integration provides a data mapping tool to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files.

3. Click **Transformation** to open the pane that allows you to define a transformation between your variable and the expected data type of the parameter on the control method.
4. In **Step 1**, click **Select Variable** to display the variables in your project. Then choose `resolutionXML` (`BugResolutionDocument`), which is the variable you created for the **Receive Resolution** node at the start of your business process.
5. In **Step 2**, click **Create Transformation**. The Transformation tool opens, which displays a representation of the BugResolution XML document in the **Source Schema** pane, and a **String** in the **Target Schema** pane.
6. Click **bug-creator** in the **Source Schema** pane and drag your mouse pointer over to **String** in the **Target Schema** pane. A line is drawn between the **bug-creator** and **String** elements in the **XML Map** pane. It represents the transformation between the two data types.



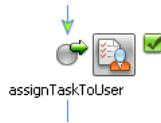
7. In the **Application** panel, double-click **ResolutionApproval.jspd** to return to the **assignTaskToUser** node builder.

Note: Creating the transformation in the preceding steps creates a Transformation control in your project: A DTF file, named **ResolutionApprovalTransformation.dtf** is created. An XQ file, which contains the query (written in the XQuery language) for the transformation method is also created. Both the DTF and XQ files are displayed in the **Application** tab. Also, an instance of the Transformation control is created and is represented as **transformations** in the **Controls** tab on the **Data Palette**.

8. In the **assignTaskToUser** node builder, click **Close**. The node builder closes.

This step completes the design of the **assignTaskToUser** node. Through this node, you specified that the person who created the bug (the **bug-creator**) is assigned the task of resolving the bug.

Step 4. Create Task and Assign to User



Note: This method sets the Assignees List to a specific Integration User: the user named in the bug-creator element of the XML document who starts the business process. Because this user is the only one on the Assignees List, this operation automatically causes the task to be claimed for the specified user.

Specify a Due Date for Completion of the Task

In the preceding step, you designed your business process to specify that the person who created the bug (the **bug-creator**) is assigned the task of resolving the bug. In this step, you specify the due date for the task. In other words, you specify the date that the user assigned to the task must complete the task.

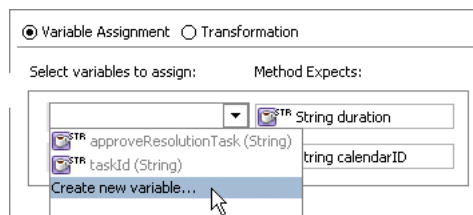
To specify a due date, you create a node on your business process by dragging a method from the **taskCtrl** in the **Data Palette** onto the business process in the same way as you created nodes in the preceding two steps.

1. In the **Data Palette**, from the **taskCtrl**, select the following method:

```
void setCompletionDueBusinessDate(String duration String calendarID)
```

Then in **Design View**, drag and drop it onto the business process, placing it on the process immediately after the **assignTaskToUser** node.

2. Double-click the **setCompletionDueBusinessDate** node, then open the **Send Data** tab. In the **Variable Assignment** pane, you can specify the variables from which your process gets the due date and the calendar name to pass to the task.
3. In the **Variable Assignment** pane, under **Select variables to assign**, in the field associated with the **String duration** parameter under **Method Expects**, click the arrow in the drop-down list and select **Create new variable...**







The **Create Variable** dialog box is displayed with Java selected as the variable type.

4. In the **Create Variable** dialog box, complete the following steps:
 - a. In the **Variable Name** field, enter `dueDate`.
 - b. In the **Default Value** field, enter `2 d` (two days). In this way, you specify the duration of business time—the time the task is due to be completed.
 - c. Select **Declare as constant**.
 - d. Click **OK**. Your new variable (of type `String`) is created and displayed in the **Send Data** tab.
5. Under **Select variables to assign**, in the field associated with the **String calendarID** parameter under **Method Expects**, click the arrow in the drop-down list and select **Create new variable...** The **Create Variable** dialog box is displayed with Java selected as the variable type.
6. In the **Create Variable** dialog box, complete the following steps:
 - a. In the **Variable Name** field, enter `calendarName`.
 - b. In the **Default Value** field, enter `SoftwareTeamCalendar`.
In this way, you specify which calendar the business process bases its calculation on. In this case, the **2 d** duration specified for the **dueDate**.
 - c. Select **Declare as constant**.
 - d. Click **OK**. Your new variable (of type `String`) is created and displayed on the **Send Data** tab.

☒ Variable Assignment ☐ Transformation

Select variables to assign: Method Expects:

 <code>dueDate (String)</code> ▼	 <code>String duration</code>
 <code>calendarName (...)</code> ▼	 <code>String calendarID</code>

7. Click **Apply**, then **Close**. The variable assignments are saved and the **setCompletionDueBusinessDate** node builder closes.

The Resolution Approval business process should now resemble the following figure:

Step 4. Create Task and Assign to User



This completes the design of a business process that creates a task, assigns the task to a user, and specifies a date by which the completion of the task is due using a Worklist control. This step also assigned the task of approving the resolution of a bug to the user who created the bug.

To design your business process to handle the other possible events for the resolution of the bug, proceed to [Chapter 6, “Step 5. Receive Resolution Approval From the Task Owner.”](#)

Create Task and Assign to User Alternative 3

In this section, you learn how to build your own Task control and customized to your needs.

This section includes the following tasks:

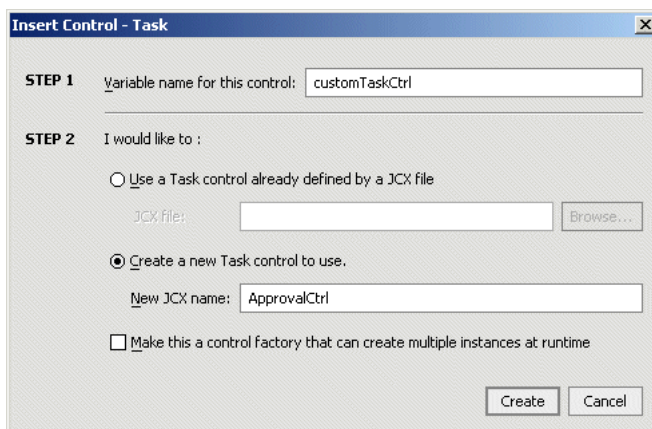
- [Create a Task Control](#)
- [Customize the Task Control](#)
- [Modify the Business Process](#)

Create a Task Control

A Task control creates and manages an instance of a Task. To create the Task control that manages the task of approving the resolution of a bug in our SoftCo enterprise, complete the following steps:

1. If the business process is not displayed in the **Design View** tab, in the **Application** panel, double-click **ResolutionApproval.jpdl**.
2. On the **Data Palette**, in the **Controls** tab, click **Add**→**Integration Controls**→**Task**. The **Insert Control - Task** dialog box is displayed.

Note: If the Controls tab is not visible in WebLogic Workshop, click **View**→**Windows**→**Data Palette** from the menu bar.



3. In **Step 1**, enter `customTaskCtrl` as the name for the instance of this control.

Step 4. Create Task and Assign to User

4. In **Step 2**, select **Create a new Task control to use**. Then, in the **New JCX name** field, enter `customTaskCtrl`.
5. Click **Create**. A new Task control and an instance of it are created and the **Insert Control** dialog box is closed.

A new JCX file, named `CustomTaskCtrl.jcx`, is created and displayed on the **Application** tab. The instance of the control is displayed on the **Controls** tab of the **Data Palette**.

Customize the Task Control

In the following steps, you customize the Task control that you created.

1. On the **Data Palette** right-click **customTaskControl**, and then select **Edit**.

The **Design View** displays all the default methods defined for this control. In this part of the tutorial, you do not need all of these methods. In the steps that follow, you will remove the unnecessary methods.



2. Click the **Source View** tab. The control's source code is displayed.
3. Remove all the methods defined in the `CustomTaskCtrl` interface except for the `Callback` interface. Your interface should look like the following:

```

/**
 * @jc:task
 */
public interface CustomTaskCtrl extends TaskControl,
    com.bea.control.ControlExtension
{
    public interface Callback extends TaskControl.Callback {
        /**
         * @jc:task-event event-type="complete" response="{response}"
         */
        void onTaskCompleted(XmlObject response);

        /**
         * @jc:task-event event-type="abort" response="{response}"
         */
        void onTaskAborted(XmlObject response);

        /**
         * @jc:task-event event-type="expire" time="{time}"
         */
        void onTaskOverdue(Date time);
    }
}

```

Note: Some methods are not customizable. If you click the **Design View** tab and look at the control instance, you will see that most of the methods have disappeared.

4. Copy the following code and replace the interface code in **Source View**.

```

public interface CustomTaskCtrl extends TaskControl,
    com.bea.control.ControlExtension
{
    public void createAndAssignTask(XmlObject requestXml, String username);
    public interface Callback extends TaskControl.Callback {
        /**
         * @jc:task-event event-type="complete" response="{response}"
         */
        void onTaskCompleted(XmlObject response);
    }
}

```

Step 4. Create Task and Assign to User

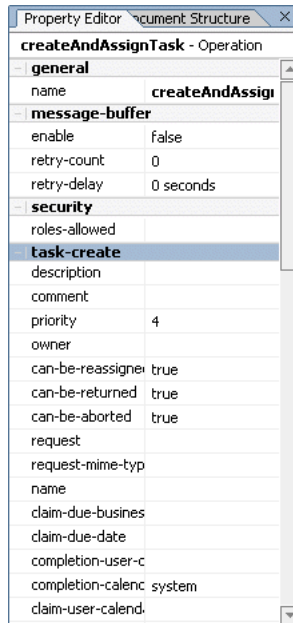
```
    /**
     * @jc:task-event event-type="abort" response="{response}"
     */
    void onTaskAborted(XmlObject response);

    /**
     * @jc:task-event event-type="expire" time="{time}"
     */
    void onTaskOverdue(Date time);
}
}
```

Note: The overloading of methods definition is not supported. When creating your own methods be careful not to use one of the default task control method names.

There are only two dynamic arguments, `request` and `assignee`, to specify at run time. In the following steps, you will define the behavior of the method and application of its arguments.

5. Click the method name (`createAndAssignTask`), and then, in the **Property Editor**, find **task-create**.
6. If necessary, expand the **task-create** property by clicking the + beside its name. The list of attributes displays all the arguments that can be supported when creating a task.



7. Specify the task name and due date, as follows:

- If necessary, click the method name (`createAndAssignTask`) in the **Source View**.
- Click the **Name** attribute inside the **task-create** property, enter `Approve Resolution`, then press **Enter**. The source code shows the following:

```
/**
 * @jc:task-create name="Approve Resolution"
 */
public void createAndAssignTask(XmlObject requestXml, String
username);
```


This method now creates a task with the name `Approve Resolution`.

- Click the **completion-due-business-date** attribute inside the **task-create** property, enter `2d`, and then press **Enter**. The source code shows the following:

```
/**
 * @jc:task-create completion-due-business-date="2d" name="Approve
Resolution"
 */
public void createAndAssignTask(XmlObject request, String username);
```


- Map the methods arguments to the task properties. This requires the use of curly brackets: `{}`.

Step 4. Create Task and Assign to User

- a. If necessary, click the method name (`createAndAssignTask`) in the **Source View**.
- b. Click the request attribute inside the **task-create** property, then click . The **Property Text Editor** is displayed.
- c. In the **Property Text Editor**, enter `{requestXml}`.

This specifies that the value passed by the `requestXml` argument is used to set the task request. To see the list of supported request types, see “Altering Method Signature—Request and Response” in [Advanced Topics](#) in *Using the Worklist*.

9. Specify the assignee, as follows:

- a. If necessary, click the method name (`createAndAssignTask`) in the **Source View**.
- b. Click the **user** attribute inside the **task-assign** property, then click . The **Property Text Editor** is displayed.
- c. In the **Property Text Editor**, enter `{username}`.
- d. Click the **algorithm** attribute inside the **task-assign** property, then click the arrow in the drop-down list and select **ToUser**. The source code shows the following:

```
/**
 * @jc:task-create request="{request}" completion-due-business-date="2d"
name="Approve Resolution"
 * @jc:task-assign user="{assignee}" algorithm="ToUser"
 */
public void createAndAssignTask(XmlObject requestXml, String
username);
```

This method now creates a task with a name Approve Resolution with a due date two business days after the task is created. The task request is passed by the `requestXml` argument and the user the task is assigned to is specified by the `username` argument.

10. In the source code, modify the method signature to return the `taskId`, by replacing `void` with `String` so it looks like the following:

```
public String createAndAssignTask(XmlObject requestXml, String
username);
```

The method will now return the `taskId`.

11. Build the application by pressing **F7**. This ensures that changes are incorporated in the JPD.

Modify the Business Process

In this section, you add a node to the Resolution Approval business process.

1. Click the **Design View** tab.
2. If the business process is not visible, in the **Application** panel, double-click the **ResolutionApproval.jpdl**.
3. Click the **Data Palette** tab.
4. On the **Data Palette**, expand the **customTaskCtrl** by clicking the + beside its name, then select **String createAndAssignTask(XMObject, String username)**. Drag and drop this method on to the business process in **Design View**, placing it on the process immediately after the **Receive Resolution** node.

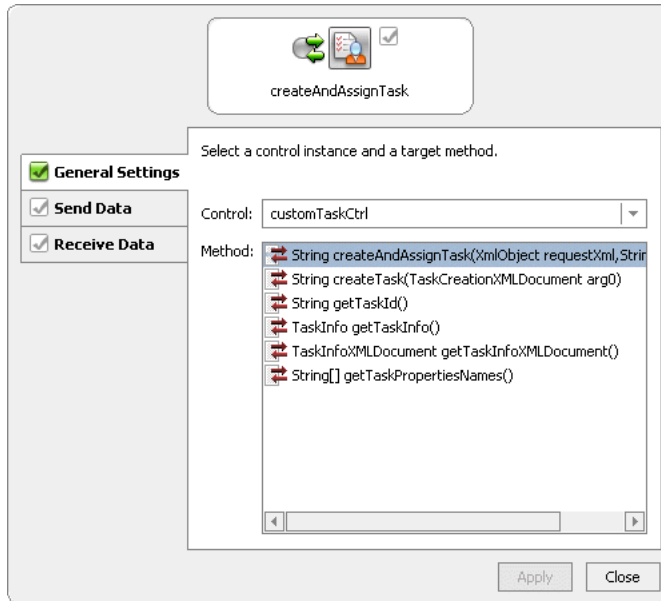


In the following steps, you will use the Transformation mode because the data type required as input to the **customTaskCtrl** control are an XML type and a Java String type. Additionally, the variable in which the Bug Resolution message is stored is of type XML (`BugResolutionDocument`). Note that the Bug Resolution message includes the names of the users to which a bug can be assigned and the `BugResolutionDocument` is valid against the `BugResolution.xsd` schema.

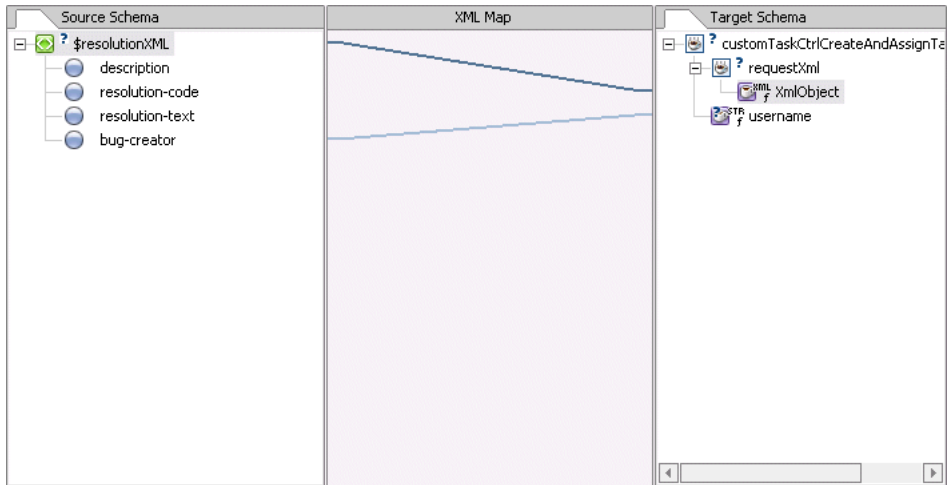
Note: WebLogic Integration provides a data mapping tool to map between heterogeneous data types. The data transformations you create using the tool are stored in Data Transformation Format (DTF) files.


Step 4. Create Task and Assign to User

5. In **Design View**, double-click the **createAndAssignTask** node to invoke the node builder. By default, the node builder opens with the **General Settings** tab selected, as shown in the following figure:



6. Open the **Send Data** tab.
7. Click **Transformation** to open the pane that allows you to define a transformation between your variable and the expected data type of the parameter on the control method.
8. In **Step 1**, click **Select Variable** to display the variables in your project. Then choose **resolutionXML (BugResolutionDocument)**, which is the variable you created for the **Receive Resolution** node at the start of your business process.
9. In **Step 2**, click **Create Transformation** to open the Transformation tool, which displays a representation of the BugResolution XML document in the Source Schema pane, and a String in the Target Schema pane.
10. Click **bug-creator** in the **Source Schema** pane and drag your mouse pointer over to **username** in the **Target Schema** pane. A line is drawn between the **bug-creator** and **username** elements in the XML Map pane.
11. Click **\$resolutionXML** in the **Source Schema** pane and drag your mouse pointer over to **XmlObject** in the **Target Schema** pane. A line is drawn between the **bug-creator** and the **XmlObject** elements in the **XML Map** pane.



12. In the **Application** panel, double-click **ResolutionApproval.jspd** to return to the **assignTaskToUser** node builder.
13. Open the **Receive Data** tab.
By default, the **Receive Data** tab opens on the **Variable Assignment** pane. The **Method Expects** field is populated with the data type expected to be returned by the `createAndAssignTask()` method: `String`.
14. Under **Select variables to assign**, click the arrow in the drop-down list, and select **Create new variable**. The **Create Variable** dialog box is displayed with Java selected as the variable type.
15. In the **Create Variable** dialog box, complete the following steps:
 - a. In the **Variable Name** field, enter `taskId`.
 - b. Click **OK**. Your new variable (of type `String`) is created and displayed in the **Receive Data** tab.
16. Click **Apply**. All tabs in the node builder should be marked complete .
17. Click **Close**. The specifications you made to the **createAndAssignTask** node are saved and the node builder closes.

This completes the creation and customizing the tutorial's Task control. In this section, you discovered the power of extending the Task control. If you want to learn more about customization, see [Advanced Topics](#) in *Using the Worklist* or look at the default Task

Step 4. Create Task and Assign to User

control that is generated when you create a new control. All the method it contains are defined using control properties and can be modified. The callback methods can also be customized.

To design your business process to handle the other possible events for the resolution of the bug, proceed to [Chapter 6, “Step 5. Receive Resolution Approval From the Task Owner.”](#)

Step 5. Receive Resolution Approval From the Task Owner

In the preceding step, you designed the logic in the business process that assigns the task of approving the resolution of a bug to the user who created the bug. The design only allowed that the task owner could approve the resolution of the bug in two days. This step describes how you design your business process to handle other possible events.

The business logic in the tutorial scenario dictates that the business process handle the receipt of one of multiple possible events from the Task control. When you design business processes in WebLogic Workshop, you can use an **Event Choice** group to create logic that causes the process to wait to receive one of a number of events. In this step, you learn how to design an **Event Choice** group for this scenario.

It includes the following tasks:

- [Create an Event Choice Group to Handle the Resolution Approval Events](#)
- [Specify the Events That the Business Process Can Receive From the Task Control](#)

Create an Event Choice Group to Handle the Resolution Approval Events


When you design business processes in WebLogic Workshop, you can use an **Event Choice** group to create logic that causes the process to wait to receive one of a number of events. After it receives one of the possible events, the flow of the business process continues. You design additional nodes within an **Event Choice** group to handle the incoming events. The flow of execution proceeds along one branch in an **Event Choice** node; the branch containing the event that happens first.

To Create an Event Choice Group

1. Ensure that the Resolution Approval business process is displayed in **Design View**.

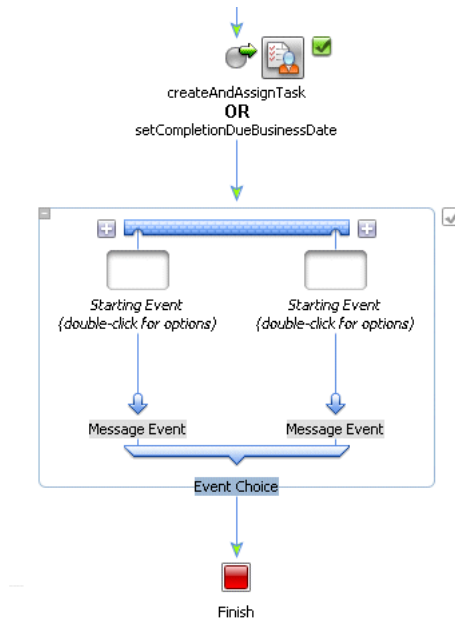
Note: You add logic to the business processes you design in WebLogic Workshop by dragging the process node that represents that logic from the **Palette** onto the process in the **Design View**.

If the **Palette** is not visible in WebLogic Workshop, choose
View→**Windows**→**Palette** from the WebLogic Workshop menu.


2. Click  **Event Choice** in the **Palette**. Then drag and drop it onto the Resolution Approval business process in **Design View**, placing it immediately after the **createAndAssignTask** (Alternatives 1, 3) or **setCompletionDueBusinessDate** (Alternative 2) node.

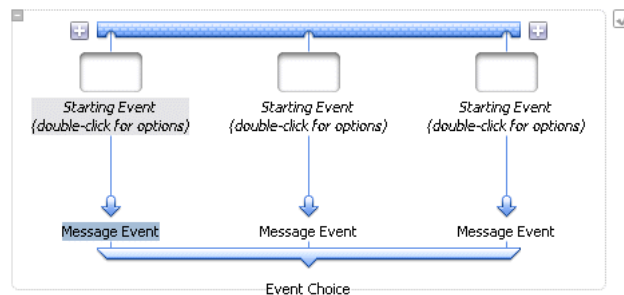
The business process is updated (in **Design View**) to contain a representation of the **Event Choice** group as shown in the following figure:

Create an Event Choice Group to Handle the Resolution Approval Events



Note that by default, **Event Choice** nodes are created with two branches. In this step, you design the Resolution Approval business process to handle a business scenario in which the task control returns one of three events. Therefore, you must expand the default **Event Choice** group to include a third **Message Event** branch, as described in the next step.

3. Click one of the  on the **Event Choice** group. An additional **Message Event** branch is created. The **Event Choice** group should resemble the following:



The **Event Choice** node can now handle the receipt of one of three possible events from the Task control.

Specify the Events That the Business Process Can Receive From the Task Control

The first node on each branch of an **Event Choice** group handles the receipt of one event. At run time, the flow of execution proceeds along one branch in an **Event Choice** group, specifically, the branch containing the event that happens first.

In the tutorial, you design the **Message Event** branches to handle the following events:

- The task is completed by the task owner; the task owner approves the resolution of the bug.
- The resolution of the bug is appealed by the task owner. Appeals must be accompanied by a document that describes the reason for the appeal.
- The task is not addressed by the task owner in the specified two day time period. In other words, the task is overdue for completion. In this scenario, bugs that are not appealed or approved in two days are assumed approved.

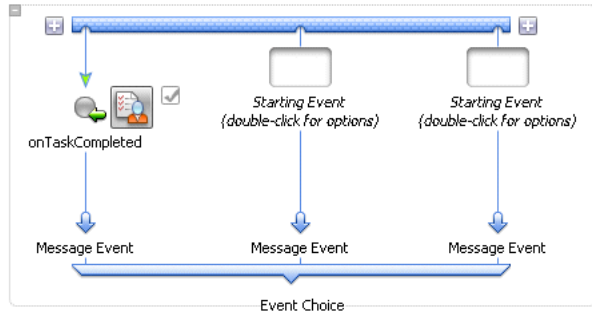
To design the **Message Event** branches, you specify a callback method on the Task control (**taskCtrl**) to create and bind to a **Control Receive** node at the beginning of each branch in the **Event Choice** group, as described in the following steps:

1. If necessary, expand the list of methods available in the **Controls** tab of the **Data Palette** by clicking the + beside the **taskCtrl** control.
2. Select the following method:

```
void onTaskCompleted(XmlObject response)
```

Then drag and drop it onto the **Event Choice** group in **Design View**, placing it on the left-most **Message Event** branch. By doing so, you specify a **Control Receive** as the **Starting Event** on the branch, and furthermore bind the **onTaskCompleted** callback

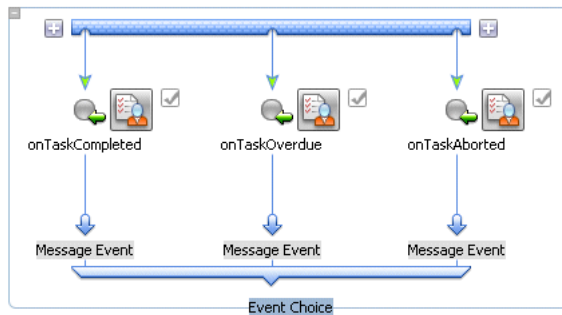
method to that **Control Receive** node. The **Event Choice** node is updated in the **Design View** to reflect the changes you made:



3. Repeat the preceding step two more times, selecting in turn the following callback methods to drag and drop from the **taskCtrl** control in the **Data Palette**:

- void onTaskOverdue(Date time)
- void onTaskAborted(XmlObject response)

After you complete this step, the **Event Choice** group in the **Design View** should resemble the following figure:



You have now completed the specification of the possible events generated by the Task control.


Design How the Callback Events are Handled by the Resolution Approval Process

At run time, the Resolution Approval business process blocks waiting for an event from the Task control before it proceeds. The flow of execution proceeds along one branch of the **Event Choice** group—the branch started by the event that happens first.

To Design For the Scenario in Which the Bug Resolution is Approved

No further design is required for the **onTaskCompleted** or **onTaskOverdue** nodes. At run time, whenever either of these message events is received, the business process exits the **Event Choice** group and proceeds to the next node in the process. The next node, which you will design in this step, is a **Client Response** node, which sends an approval message to the client. This node is followed by the **Finish** node, which marks the end of the business process.

To create the **Client Response** node, complete the following steps:

1. In the **Palette**, click  **Client Response**. Then drag and drop it onto the Request Approval business process in **Design View**, placing it on the process immediately after the **Event Choice** group and immediately before the **Finish** node.
2. Rename the node from **Client Response** to **Resolution Approved**.
3. Double-click the **Resolution Approved** node to open its node builder.
4. On the **General Settings** tab, in the **Method Name** field, enter `approved`.
5. Click **Apply**, then **Close**.

No further specifications are required on the **Resolution Approved** node. The business logic in the tutorial scenario does not require that a document is sent to the client as the result of the bug resolution being approved.

To Design For the Scenario in Which the Bug Resolution is Appealed

However, because the SoftCo business scenario dictates that appeals must be accompanied by a document that describes the reason for the appeal, we want to design additional logic in the process for the case in which an **onTaskAborted** message event is received. This section describes how to design the **onTaskAborted** branch of the **Event Choice** group to define the flow of execution when the Task owner appeals the resolution of the bug. This section contains the following steps:

- [To Design the onTaskAborted Message Event Branch](#)
- [To Design the Client Response in the Case of an onTaskAborted Message](#)

To Design the onTaskAborted Message Event Branch

1. Double-click the **onTaskAborted** node to open its node builder. The node builder opens on the **General Settings** tab. The Control instance and target methods are already selected:
 - Alternatives 1, 2—**TaskCtrl** and **void onTaskAborted(XmlObjectResponse)**

- Alternative 3—**customTaskCtrl** and **void onTaskAborted(XmlObject response)**

2. Open the **Receive Data** tab.

☒ Variable Assignment ☐ Transformation


Select variables to assign: Method Expects:

3. Create a variable to which the document returned by the Task control, containing the reason for the appeal, is assigned at run time:
 - a. Under **Select variables to assign**, click the arrow in the drop-down list and select **Create new variable...** The **Create Variable** dialog box is displayed with XML selected as the variable type.
 - b. In the **Variable Name** field, enter `appealXML`.
 - c. Click **OK**. Your new variable (of type XmlObject) is created and displayed in the **Receive Data** tab.
4. On the node builder, click **Apply**, then **Close**. The specifications you made on this node are saved and the node builder closes.

At run time, the document returned by the task owner that describes the reasons for appealing the resolution of the bug is assigned to the **appealXML** variable.

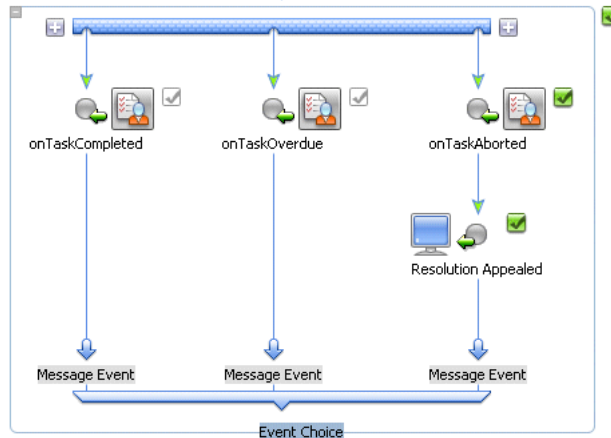
The last step in designing the logic for the SoftCo business scenario is to define a node in the business process that returns the document containing the reasons for the appeal to the client. Note that the client is the same resource that starts the Resolution Approval business process.

To Design the Client Response in the Case of an **onTaskAborted** Message

1. Click  **Client Response** in the **Palette**. Then in **Design View**, drag and drop it onto the Request Approval business process, placing it on the process immediately after the **onTaskAborted** node in the **Event Choice** group.
2. Rename the node from **Client Response** to **Resolution Appealed**.

The business process is updated in the **Design View**, as shown in the following figure:

Step 5. Receive Resolution Approval From the Task Owner



3. Double-click the **Resolution Appealed** node in the **Event Choice** group to open its node builder.
4. On the **General Settings** tab, in the **Method Name** field, enter `appealed`.
 - a. Click **Add**. A panel showing the data types you can use is displayed:

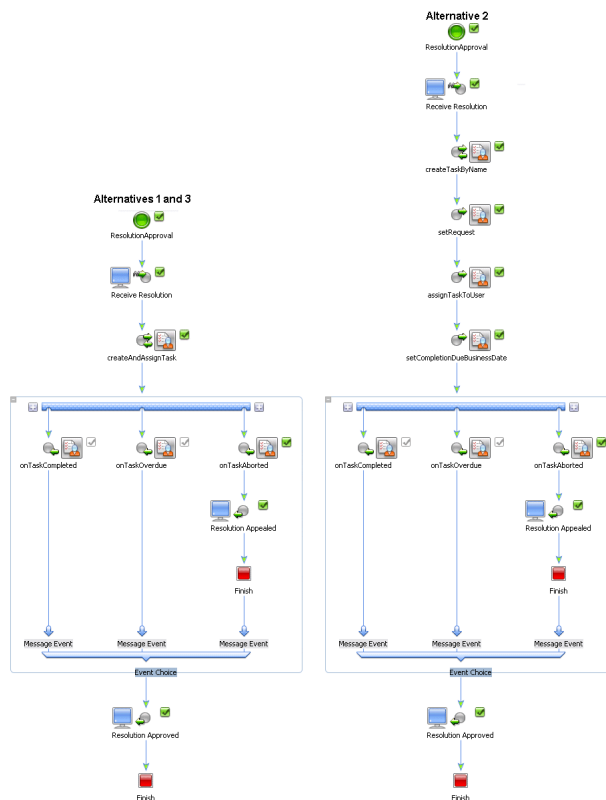
The appeal message returned to the business process from the Task control in the preceding node (**onTaskAborted**) is an (untyped) XML message. Therefore you need to use **XML** for this node.
 - b. If necessary, select **XML**.
 - c. Click the **+** beside **Untyped** to expand the list of *Untyped XML* objects available in your project.
 - d. Select **XmlObject**.
 - e. In the **Name** field, replace the default parameter name (**x0**) with **appealXML**.
 - f. Click **OK**. The parameter specifications you made are displayed in **General Settings** tab of the node builder. The parameter type and name are **XmlObject** and **appealXML** respectively.
5. Click the **Send Data** tab. The **Send Data** tab opens and displayed the **Variable Assignment** pane by default.

6. Under **Select variables to assign**, select **appealXML (XMLObject)** from the drop-down list.
7. Click **Apply**, then **Close**.

This step completes the specification of the callback method exposed to clients by the Resolution Approval business process.

8. On the **onTaskAborted** branch, add a **Finish** node immediately after the **Resolution Appealed** node to specify that at run time the business process is terminated immediately after the **appealXML** document is sent to the client that invoked the business process.

In the **Design View**, the completed Resolution Approval process should resemble the following figure.



9. Select **File→Save All**. Your work is saved.

Step 5. Receive Resolution Approval From the Task Owner

This completes the design of the **Event Choice** group, and the Resolution Approval business process. To run and test the functionality of the business process you created using WebLogic Workshop's browser-based interface, proceed to [Chapter 7, “Step 6. Run the Resolution Approval Business Process.”](#)

Step 6. Run the Resolution Approval Business Process

You can run and test the functionality of the business process you created using WebLogic Workshop's browser-based interface. Using the **Workshop Test Browser**, you play the role of the client, invoking the methods on the business process and viewing the responses.

You also use the Worklist user interface (UI) to play the role of QualityEngineerA at the SoftCo enterprise. This step describes how to test two scenarios; one in which QualityEngineerA approves the resolution of the bug, one in which QualityEngineerA appeals the resolution of the bug.

To Launch the Test Browser


1. In the **Application** pane, select **ResoutionApproval.jpdl**—the business process you want to test.
2. If it not already selected, select the **Design View** tab. The business process you selected in the **Application** pane is displayed in **Design View**.
3. If WebLogic Server is running in the `worktutorial` domain, the following indicator is visible in the status bar at the bottom of the WebLogic Workshop visual development environment:

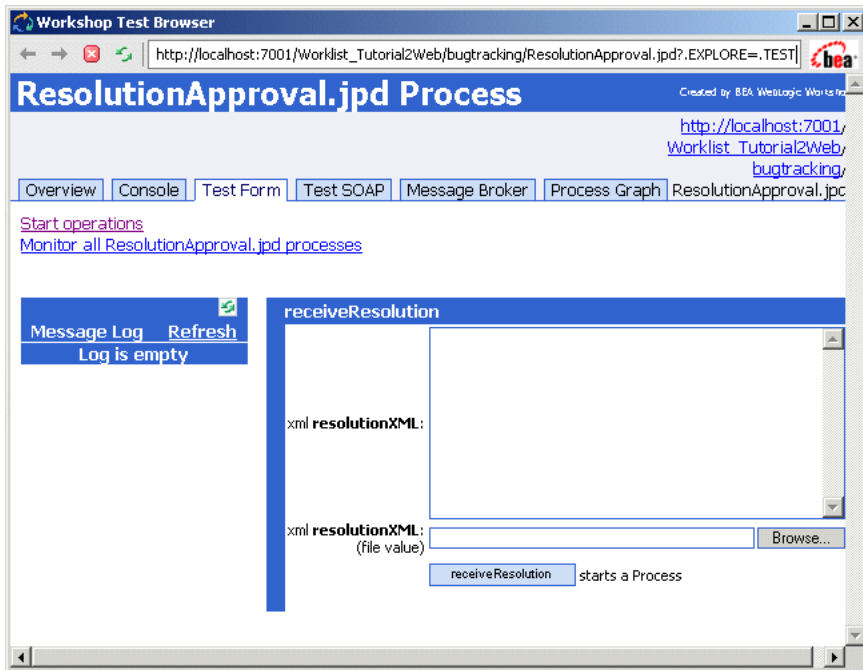


If the server is not already running, from the WebLogic Workshop menu, start WebLogic Server. To do so, choose **Tools**→**WebLogic Server**→**Start WebLogic Server**.

4. Build the application by pressing **F7**, or from the WebLogic Workshop menu, click **Build**→**Build Application**. WebLogic Workshop builds your application.

Step 6. Run the Resolution Approval Business Process

5. After the build is complete and the server is updated, click the Start button  on the menu bar to run your business process. The **Workshop Test Browser** is launched, through which you can test your business process using sample input values.



6. In the **Test Browser**, if necessary, click the **Test Form** tab to open the **Test Form** page.
The **Test Form** page allows you to enter data that your business process can receive as part of a client request.
7. Select the XML data provided in [Listing 7-1](#), then copy it and paste it into the box labeled **xml resolutionXML** on the Test Form page.

Listing 7-1 sampleFixResolution.xml

```
<bug:bug-resolution
  xmlns:bug="http://www.bea.com/WLIWorklistTutorial/BugResolution.xsd">
  <bug:description>Null Pointer Exception while performing
  XYZ</bug:description>
```

```

    <bug:resolution-code>FIXED</bug:resolution-code>

    <bug:resolution-text>I forget to check for null. Now we use a default
in that case.</bug:resolution-text>

    <bug:bug-creator>QualityEngineerA</bug:bug-creator>
</bug:bug-resolution>

```

8. In the **Test Browser**, click the button labeled with the method name on your business process (**receiveResolution**) to invoke the method. The **Test Form** page refreshes to display a summary of your request parameters and the calls in the **Message Log**.

The first call is logged in the Message Log. It is the **receiveResolution** call you made to invoke the business process.

To Launch the Worklist User Interface

To drive the further execution of the business process, you must log into the Worklist UI and play the role of **QualityEngineerA**—the person who created the bug. See the **bug-creator** element in **sampleFixResolution.xml**, which is the test XML used to invoke the business process and one of the users you set up for the tutorial scenario.

1. Log on to the Worklist UI in one of the following ways:
 - From the WebLogic Workshop menu:
Tools→WebLogic Integration→Worklist
 - Enter the following URL in a Web browser:
`http://localhost:7001/worklist`
2. Enter `QualityEngineerA` for the username and use the password you set up for `QualityEngineerA` in “[Configure the Users and Groups](#)” on page 2-3.
3. Click **Claimed Tasks** to see the task. This task was assigned directly to `QualityEngineerA`, who claims it automatically.

QualityEngineerA Logout		Worklist					
Views		Task	Description	DueDate	Priority	State	Owner Claimant
Owned Tasks		Approve					
Assigned Tasks		Resolution		11/3/03 10:00 AM	1	claimed	<anonymous> QualityEngineerA details
Claimed Tasks							
Associated Tasks							

4. Click **details** to open a page that displays the details of the claimed task.

Step 6. Run the Resolution Approval Business Process

Task details	
start	return
abort	response
Details	
Task Name	Approve Resolution
Task Description	
Task ID	192.168.11.15-1f15f7f.f65b379381.-7fec
Parent Process	/Worklist_Tutorial/Veb/bugtracking/ResolutionApproval.jspd
Parent Process ID	1058093686973
State	claimed
Creation Date	7/13/03 3:54 AM
Claim Due Date	
Due Date	7/16/03 9:00 AM
Task Assignee	QualityEngineerA
Task Claimant	QualityEngineerA
Task Owner	<anonymous>

5. Click **response** to open a page that displays the XML describing the resolution (the RequestXML).

Response	
Task Response	
RequestXML	<pre><bug:bug-resolution xmlns:bug="http://www.bea.com/MWLWorklistTutorial/BugResolution.xsd"> <bug:description>Null Pointer Exception while performing XYZ</bug:description> <bug:resolution-code>FIXED</bug:resolution-code> <bug:resolution-text>I forget to check for null. Now we use a default in that case.</bug:resolution-text> <bug:bug-creator>QualityEngineerA</bug:bug-creator> </bug:bug-resolution></pre>
ResponseXML	
<div>OkCancel</div>	

6. Click the **Back** button in your browser to go back to the **Task details** page.
7. On the **Task details** page, click **start**, then **complete**. In this way you complete the task to approve the resolution.

Task details

[response](#)

Details

Task Name	Approve Resolution
Task Description	
Task ID	10.61.4.154-4307f2.f7b0477dfd.-7ff6
Parent Process	/Worklist_Tutorial2Web/bugtracking/ResolutionApproval.jsp
Parent Process ID	1063986667499
State	completed
Creation Date	9/19/03 9:51 AM
Claim Due Date	
Due Date	9/22/03 10:00 AM
Task Assignee	QualityEngineerA
Task Claimant	QualityEngineerA
Task Owner	<anonymous>

- Return to the **Workshop Test Browser** and click **Refresh** on the **Message Log** pane. Note that the business process runs to completion.

Message Log

Refresh

1063986667499

Monitor

Graph

→ receiveResolution

taskCtrl:listener.onTaskEvent ←

← callback.approved

Instance 1063986667499 is Completed.

Clear Log

Service Request receiveResolution

Submitted at Fri Sep 19 09:51:07 MDT 2003

```

.CONVPHASE = .START
resolutionXML = <bug:bug-resolution
xmlns:bug="http://www.bea.com/WLIWorklistTutorial/BugResolution.xsd">
<bug:description>Null Pointer Exception while performing
XYZ </bug:description>
<bug:resolution-code>FIXED </bug:resolution-code>
<bug:resolution-text>I forget to check for null. Now we use a default in that
case. </bug:resolution-text>
<bug:bug-creator>QualityEngineerA </bug:bug-creator>
</bug:bug-resolution>
.CONVERSATIONID = _ID_
.EXECREQUEST = true

```

- Click **← callback.approved**. The **Message Log** resembles the following figure:

Message Log

Refresh

1058093686973

Monitor

Graph

→ receiveResolution

taskCtrl:listener.onTaskEvent ←

← callback.approved

Instance 1058093686973 is Completed.

Clear Log

Client Callback

Submitted at Sun Jul 13 04:12:07 PDT 2003

approved()

No Response

Submitted at Sun Jul 13 04:12:07 PDT 2003

The original client is the Test User Interface

The following calls are logged in the **Message Log**:

- receiveResolution**—the call you made (acting in the role of the client) to invoke the business process.
- taskCtrl:listener.onTaskEvent**—the call from the Task Control to the Resolution Approval business process.


Step 6. Run the Resolution Approval Business Process

- **callback.approved**—the callback to the client that invoked the business process. In this case the method is the **approved** method you created on the **Resolution Approved** node in the business process.

Congratulations! You have successfully completed running and testing the Resolution Approval business process for a scenario in which the user (QualityEngineerA) approves the resolution of the bug.

The following steps describe the data and steps you need to run the business process again, but this time, QualityEngineerA appeals the task—that is, QualityEngineerA appeals the resolution of the bug.

Run ResolutionApproval.jpdl Again

1. Close the **Test Browser**, then run the **ResolutionApproval.jpdl** business process again by clicking the Start button  on the menu bar in WebLogic Workshop.
2. Select the XML data provided in **Listing 7-1, “sampleFixResolution.xml,” on page 7-2**, then copy it and paste it into the box labeled **xml resolutionXML** on the Test Form page.
3. Click the button labeled with the method name on your business process (**receiveResolution**) to invoke the method. The first call is logged in the Message Log. It is the **receiveResolution** call you made to invoke the business process.
4. Access the Worklist UI, again playing the role of QualityEngineerA, and once again select **Claimed Tasks**.
5. Click **details** to open a page that displays the details of the claimed task.

Note: On the **Details** page, note the values in the **Creation Date** and the **Due Date** fields. Note that the owner of the task (QualityEngineerA) has two business days in which to accept or appeal the resolution of the software bug.

Creation Date	9/19/03 2:14 PM
Claim Due Date	
Due Date	9/22/03 2:15 PM

6. Click **response** to open a page that displays the XML describing the resolution (the RequestXML).

Note: Recall that the SoftCo enterprise requires that users who appeal the resolution of a bug attach a document that describes the reason for their appeal. In this scenario, QualityEngineerA appeals the task and includes the document that describes why. The following steps describe how to test this scenario.

7. Select the XML data provided in [Listing 7-2](#), then copy it and paste it into the **ResponseXML** box on the **Task Response** page, as shown in [Figure 7-1](#).

Listing 7-2 sampleAppeal.xml

```
<bug:resolution-appeal
  xmlns:bug="http://www.bea.com/WLIWorklistTutorial/ResolutionAppeal.xsd">
  <bug:appeal-text>Please reconsider, a customer has
complained.</bug:appeal-text>
  <bug:resolution-code>WILL_NOT_FIX</bug:resolution-code>
  <bug:resolution-text>This is not important, we have other bugs to
fix.</bug:resolution-text>
</bug:resolution-appeal>
```

Figure 7-1 ResponseXML box

The screenshot shows a dialog box titled "Task Response". It contains two text areas: "RequestXML" and "ResponseXML". The "RequestXML" area contains the following XML code:

```
<bug:bug-resolution xmlns:bug="http://www.bea.com/WLIWorklistTutorial/BugResolution.xsd">
  <bug:description>Null Pointer Exception while performing XYZ</bug:description>
  <bug:resolution-code>FIXED</bug:resolution-code>
  <bug:resolution-text>I forget to check for null. Now we use a default in that case.</bug:resolution-text>
  <bug:bug-creator>QualityEngineerA</bug:bug-creator>
</bug:bug-resolution>
```

The "ResponseXML" area contains the following XML code:

```
<bug:resolution-appeal
  xmlns:bug="http://www.bea.com/WLIWorklistTutorial/ResolutionAppeal.xsd">
  <bug:appeal-text>Please reconsider, a customer has complained.</bug:appeal-text>
  <bug:resolution-code>WILL_NOT_FIX</bug:resolution-code>
  <bug:resolution-text>This is not important, we have other bugs to fix.</bug:resolution-text>
</bug:resolution-appeal>
```

At the bottom of the dialog box are two buttons: "Ok" and "Cancel".

8. Click **Ok**. The browser returns to the **Task details** page.
9. On the **Task details** page, click **start**, then **abort**. In this way you abort the task to appeal the resolution.
10. Return to the **Workshop Test Browser** and click **Refresh** on the **Message Log** pane. Note that the business process runs to completion.

Run the Resolution Approval Business Process

Message Log

Refresh

1058096347909

Monitor
 Graph

receiveResolution
 taskCtrlListener.onTaskEvent

callback.appealed
 Instance **1058096347909** is Completed.

Clear Log

Client Callback

Submitted at Sun Jul 13 04:57:39 PDT 2003
 Appealed<bug:resolution-appeal xmlns:bug="http://www.bea.com/WLIWorklistTutorial/ResolutionAppeal.xsd"><bug:appeal-text>Please reconsider, a customer has complained.</bug:appeal-text><bug:resolution-code>WILL_NOT_FIX</bug:resolution-code><bug:resolution-text>This is not important, we have other bugs to fix.</bug:resolution-text></bug:resolution-appeal>

No Response

Submitted at Sun Jul 13 04:57:39 PDT 2003
 The original client is the Test User Interface

Note that the final callback logged in the Message Log is **callback.appealed** (the callback to the client that invoked the business process). The method is the **appealed** method you created on the **Resolution Appealed** node in the **onTaskAborted** Message Event branch of the business process.

This completes running and testing the Resolution Approval business process for a scenario in which the user (QualityEngineerA) appeals the resolution of the bug.