BEA WebLogic Workshop™ Help

Version 8.1 SP4
December 2004

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Table of Contents

Getting Started **1**

Logging In to the Avitek Corporate Intranet **2**

Viewing Employee Information **47**

Ordering Office Equipment **54**

Learning More About WebLogic Platform **78**

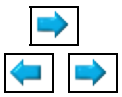
Getting Started

Welcome to the BEA WebLogic Platform Tour!

The BEA WebLogic Platform Tour introduces you to the main features of BEA WebLogic Platform and provides a launching point to get you started developing and integrating your own applications.

To get started using the WebLogic Platform Tour, review the following sections:

- Introducing the WebLogic Platform Tour
- Stepping Through the WebLogic Platform Tour

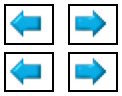


Logging In to the Avitek Corporate Intranet

The Tour begins here! In this section you (new employee John Smith) log in to the Avitek corporate intranet. This section describes the login process and the features of the intranet portal, demonstrating the integration of an application built using WebLogic Portal. (WebLogic Portal is delivered as part of the WebLogic Workshop Platform Edition.)

Follow the steps described in Step Through the Tour and review the concepts described in this section to learn about:

- Developing an Intranet Portal Using WebLogic Portal
- Reviewing the Avitek Intranet Portal
- Building a Custom Control
- Reviewing the Custom Controls for the Avitek Intranet Portal
- Configuring Security in Web Applications



Step Through the Tour

When the WebLogic Platform Tour Web application is launched, you are prompted to log in to the Avitek intranet portal.

Begin by logging in to the Avitek corporate intranet as new employee John Smith:

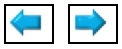
1. Enter the following user name and password in the Avitek Log In window:

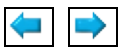
   User Name: **john**

   Password: **employee**

2. Click **Log In**.

The employee view of the intranet portal is displayed.





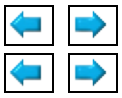
Developing an Intranet Portal Using WebLogic Portal

WebLogic Portal simplifies the development, management, and delivery of portals. Using WebLogic Workshop with the WebLogic Workshop Portal Extensions, you can graphically design portals, integrating web services, Web applications, and business processes. Once you build portals using WebLogic Workshop, you can deploy and manage them using WebLogic Portal.

The following sections describe the main features of WebLogic Portal:

- Designing a Portal

- Designing a Portlet
- Controlling the User Path Using Page Flows
- Creating JSP Content
- Administering Portals With WebLogic Portal



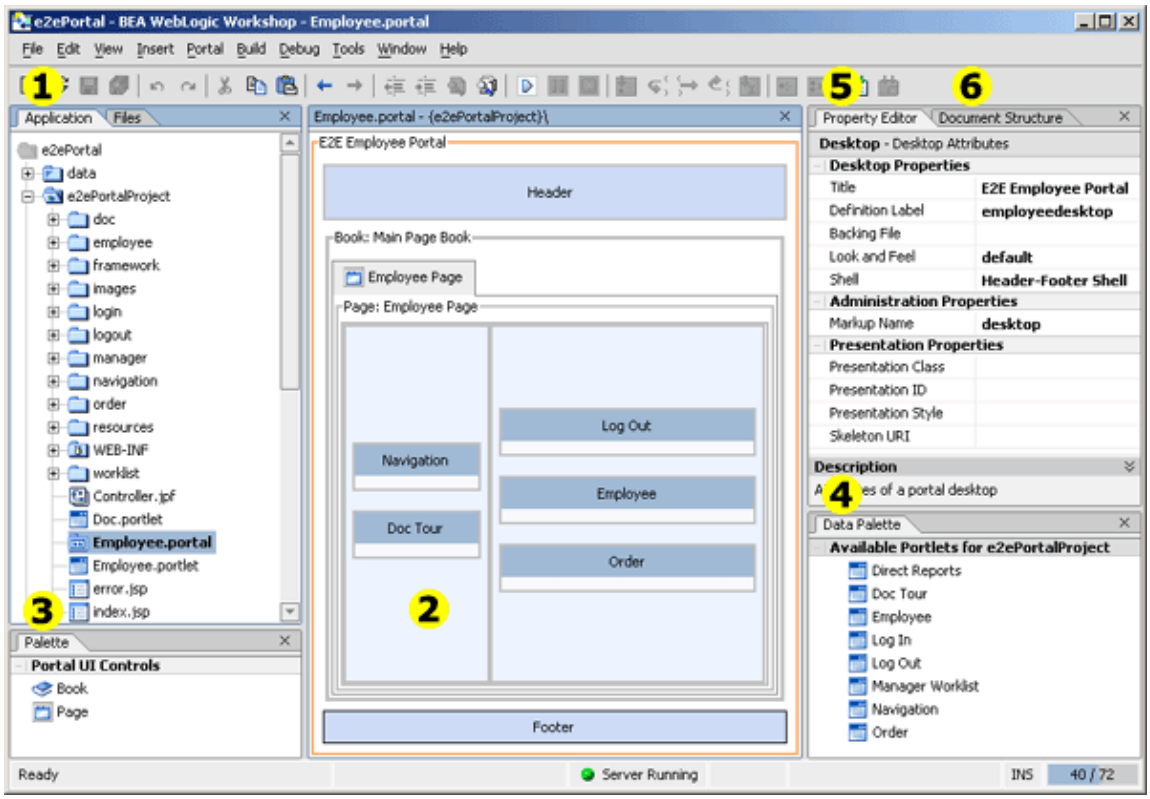
Designing a Portal

A portal provides a user interface for Web−based applications. It is a single point of access to enterprise data and applications, presenting a unified and personalized view of that information to employees, customers, and business partners.

WebLogic Workshop makes it easy to create a portal: click the target folder and choose File > New > Portal. The Portal Designer opens and displays a portal with a default header, body, and footer.

The Portal Designer is a graphic tool that enables you to design portals such as the Employee portal shown in the following figure.

Portal Designer



The following table describes the portal design tools shown in the previous figure.

Tools for Designing Portals

Callout #
Use this tool...
For these tasks...

1

Application Window

Create, view, and edit portal files in your portal application projects. The names of portal files end in .portal.

The Portlet Designer stores the portal as an XML file. You can view the portal file in XML format within WebLogic Workshop by **right−clicking** the portal file and selecting:

- **External Tools** > **XMLSpy** if you installed this tool.
- **Open as XML** from the drop−down menu when the Open as XML portal application property is enabled.

To enable the Open as XML portal application property:

1. Choose **Tools** > **Application Properties** to open the Application Properties window.
2. Select the **Portal** folder in the left−hand pane.
3. Select the **Is the Open as XML Option available?** Portal option by clicking its checkbox.
4. Select **OK** to apply the changes and close the Application Properties window.

2

Design View

Design your portal in this area.

3

Palette Window

Add books and pages to the portal by dragging components from the Palette window and dropping them onto the Design View canvas.

4

Data Palette Window

Add portlets to the portal by dragging them from the Data Palette window and dropping them onto the Design View canvas. You can use the sample portlets provided by WebLogic Portal, or you can create your own, as described later in Designing a Portlet.

5

Property Editor Window

Set properties for the portal component that is currently selected, such as the portal look and feel. You can select a portal component by clicking on it in the Design View canvas or by selecting its name in the Document Structure window (described below).

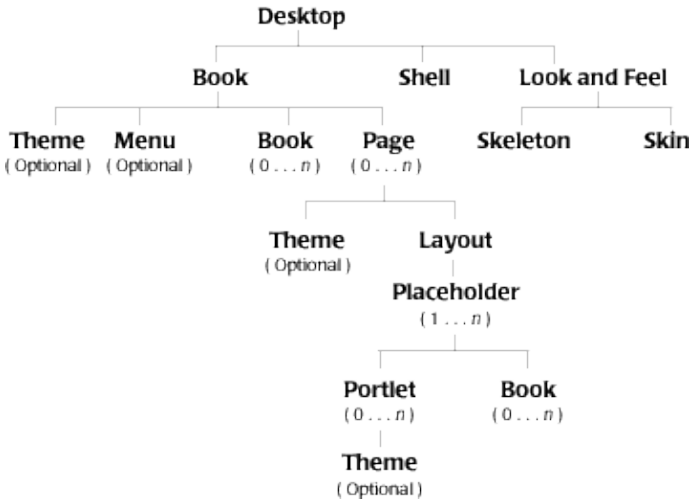
Document Structure Window

View the components of the portal interface in a hierarchical structure (see Figure: Document Structure Window for a Portal).

You can select a portal component in the Document Structure window by clicking on it. The portal component is selected in Design View and you can edit its properties in the Property Editor window.

The following figure shows the hierarchical structure of a portal.

Hierarchical Structure of a Portal



As shown in the previous figure, the following components make up the hierarchical structure of a portal:

- Desktop User view of portal components. The desktop component is created when you add a portal to the project.
- Book High−level content organization and navigation for the content; provides a mechanism for nesting pages and other content. A single book is added when you create a portal. Include additional books using the Palette window.
- Menu Scheme for navigating among books and pages. To define the menu navigation for a Book component, select a navigation scheme in the Navigation field in its Property Editor window.
- Page Organized collection of portlets and books whose position is determined by the layout. A single page is added when you create a portal. Include additional pages using the Palette window.
- Layout Position of the portlets and books on a page. Layouts define the placeholders (rows and columns) in which portlets can be placed. They also define whether books and portlets are placed on top of or beside one another within a placeholder. Layouts use the skeleton (described later) defined by the Desktop to render its content. To define the layout for a Page component, select a layout in the Layout Type field in its Property Editor window.
- Placeholder Individual cells in the layout used to organize the portlets on a page. Placeholders are added based on the layout type selected.
- Portlet Window in which you present your applications, information, and business processes. Portlets are described in detail in the next section. Add portlets using the Data Palette.
- Shell Rendered area surrounding a portal desktop's main content area (books, pages, and portlets). The shell controls the content that appears in a desktop's header and footer regions. To configure a

shell to use specific JSPs or HTML files to display content especially personalized content in a header or footer, select a shell in the Shell field in its Property Editor window.

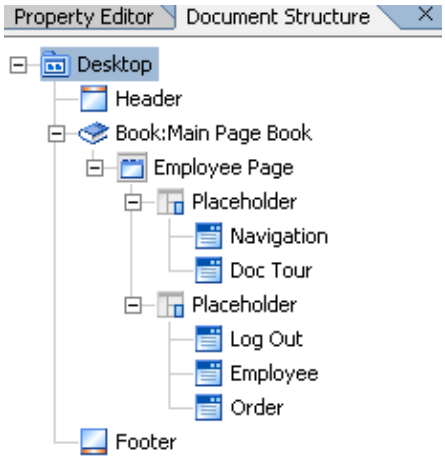
- Look and Feel Stylistic aspects of the portal. A look−and−feel definition contains the following elements:
  - ♦ Skeletons JSP files that define the physical boundaries of portal components such as header, footer, book, page, portlet, and portlet title bar.
  - ♦ Skins Collections of graphics and cascading style sheets (CSS) that control the stylistic elements of a portal, such as colors and graphics, and keep these elements separate from the portal content. Using skins, you can modify the look and feel of your portal without changing any portal components.

    A portal look and feel consists of a single XML file (with a .laf extension) that points to the skeleton and skin definitions. To define the look−and−feel definition for the Desktop component, select one in the Look and Feel field in its Property Editor window.

- Theme Subset of skins designed for books, pages, and portlets. Themes provide a way of using a different set of styles for individual desktop components. Once the look and feel is defined, to define a theme for a Book Page, or Portlet, select a theme in the Theme field in its Property Editor window.

The Document Structure window shows the relationship between specific components in the current portal file.

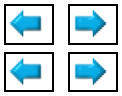
Document Structure Window for a Portal



Note that the Header and Footer components in the Document Structure window represent areas above and below the main body that typically include personalized content, banner graphics, legal notices, and related links.

To learn more about creating and designing portals:

- View a demo, Designing a Portal, that shows how to use the portal design tools.
- See Portal Tutorials in the *WebLogic Workshop Help*.
- See Building Portal Applications in the *WebLogic Workshop Help*.
- See WebLogic Administration Portal Online Help, available on E−docs, for information about managing portal applications.



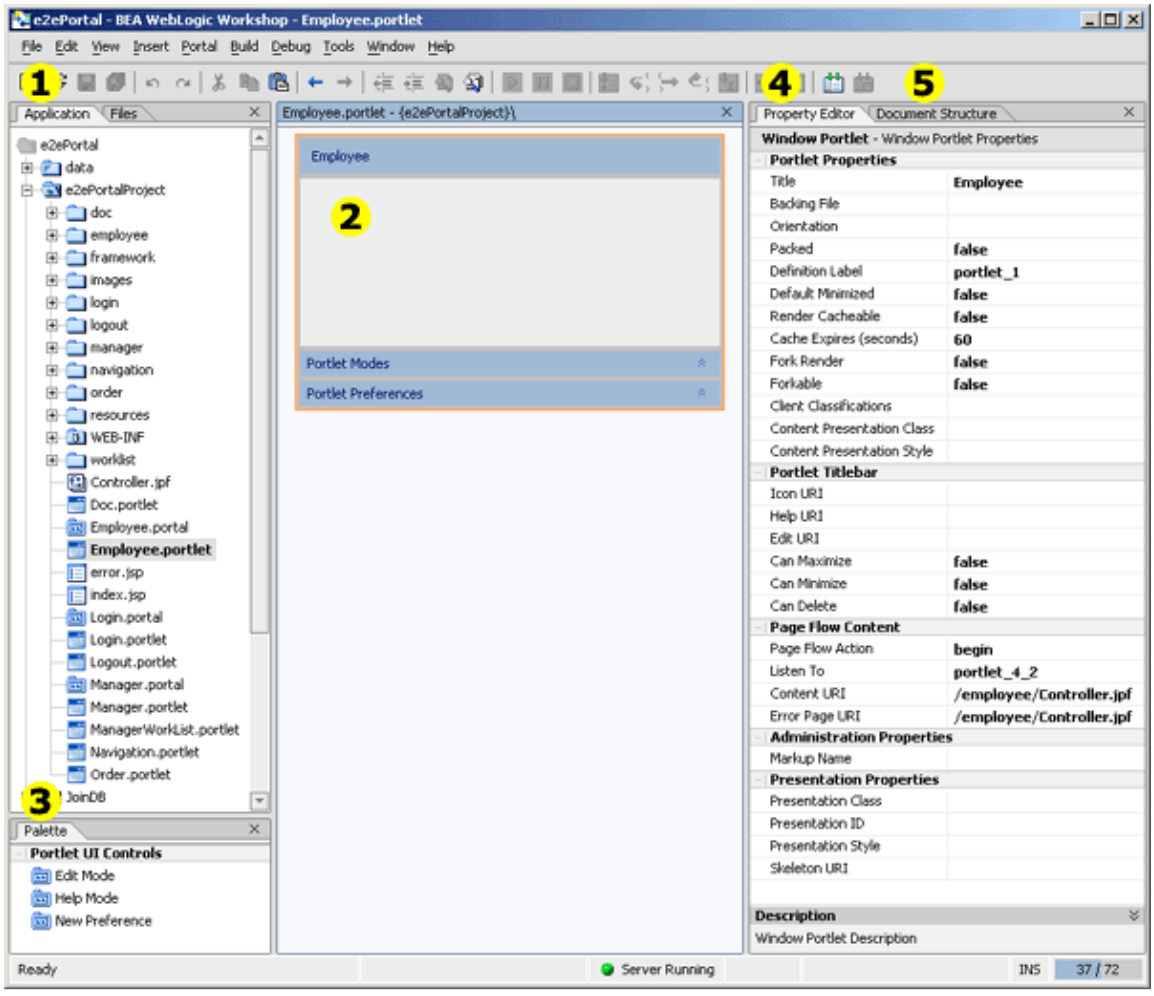
Designing a Portlet

Each page within a portal can contains a set of nested pages, or *portlets* user−specific resources on a portal page. Portlets provide access to specific applications or services, giving users access to multiple sources of information, business processes, and applications in a single place.

WebLogic Workshop makes it easy to create a portlet: click the target folder and choose File > New > Portlet. The Portlet Wizard steps you through the process of creating an initial portlet and opens it in the Portlet Designer.

The Portlet Designer is a graphic tool that enables you to design portlets such as the Employee portlet shown in the following figure. The Portlet Designer stores the portlet as an XML file.

Portlet Designer



The following table describes the portlet design tools shown in the previous figure.

Tools for Designing Portlets

Callout #
Use this tool...
For these tasks...

1

Application Window

Create, view, and edit portlet files in your portal application projects. The names of portlet files end in .portlet.

The Portlet Designer stores the portlet as an XML file. You can view the portlet file in XML format within WebLogic Workshop by **right−clicking** the portlet file and selecting:

- **External Tools** > **XMLSpy** if you installed this tool.
- **Open as XML** from the drop−down menu when the Open as XML portal application property is enabled.

To enable the Open as XML portal application property:

1. Choose **Tools** > **Application Properties** to open the Application Properties window.
2. Select the **Portal** folder in the left−hand pane.
3. Select the **Is the Open as XML Option available?** Portal option by clicking its checkbox.
4. Select **OK** to apply the changes and close the Application Properties window.

2

Design View

Design your portlet in this area.

3

Palette Window

Add one of the following controls to the portlet:

- Help Mode Adds an icon to the titlebar. Users click the icon to access help information. You set the associated Help URI value in the Property Editor window.
- Edit Mode Adds an icon to the titlebar. Users click the icon to load a page that enables them to edit the contents of the current portlet. You set the associated Edit URI value in the Property Editor window to define the page that is loaded when the user clicks the Edit icon.
- New Preference Adds customizable application properties at the developer level that can be exposed to the administrator and end user.

4

Property Editor Window

Set properties for the portlet component that is currently selected. For example, you can set the Content URI property to identify the page flow or JSP that defines the initial content of the portlet. You can select a portlet component by clicking on it in the Design View canvas or by selecting its name in the Document Structure window (described below).

5

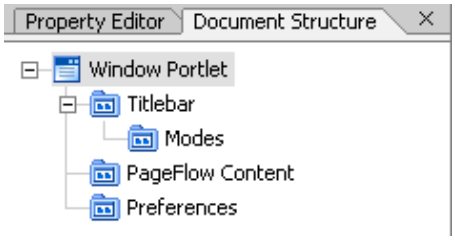
Document Structure Window

View the components of the portlet interface in a hierarchical structure (see Figure: Document Structure Window for a Portlet).

You can select a portlet component in the Document Structure window by clicking on it. The portal component is selected in Design View and you can edit its properties in the Property Editor window.

The following figure shows the Document Structure window for a portlet.

Document Structure Window for a Portlet

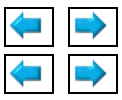


As shown in the Document Structure window, the following components make up the hierarchical structure of the sample Employee portlet interface:

- Titlebar Defines the contents of the heading displayed (if enabled) above the portlet. Contents may include Icon, Help, and Edit URI values, and options for maximizing, minimizing, or deleting the portlet.
- Page Flow Content Defines the page flow content of the portlet.
- Preferences Defines custom preferences.

To learn more about designing portlets:

- View a demo, Designing a Portlet, that shows how to use the portlet design tools.
- See Portal Tutorials in the *WebLogic Workshop Help*.
- See Creating Portlets in the *WebLogic Workshop Help*.



Controlling the User Path Using Page Flows

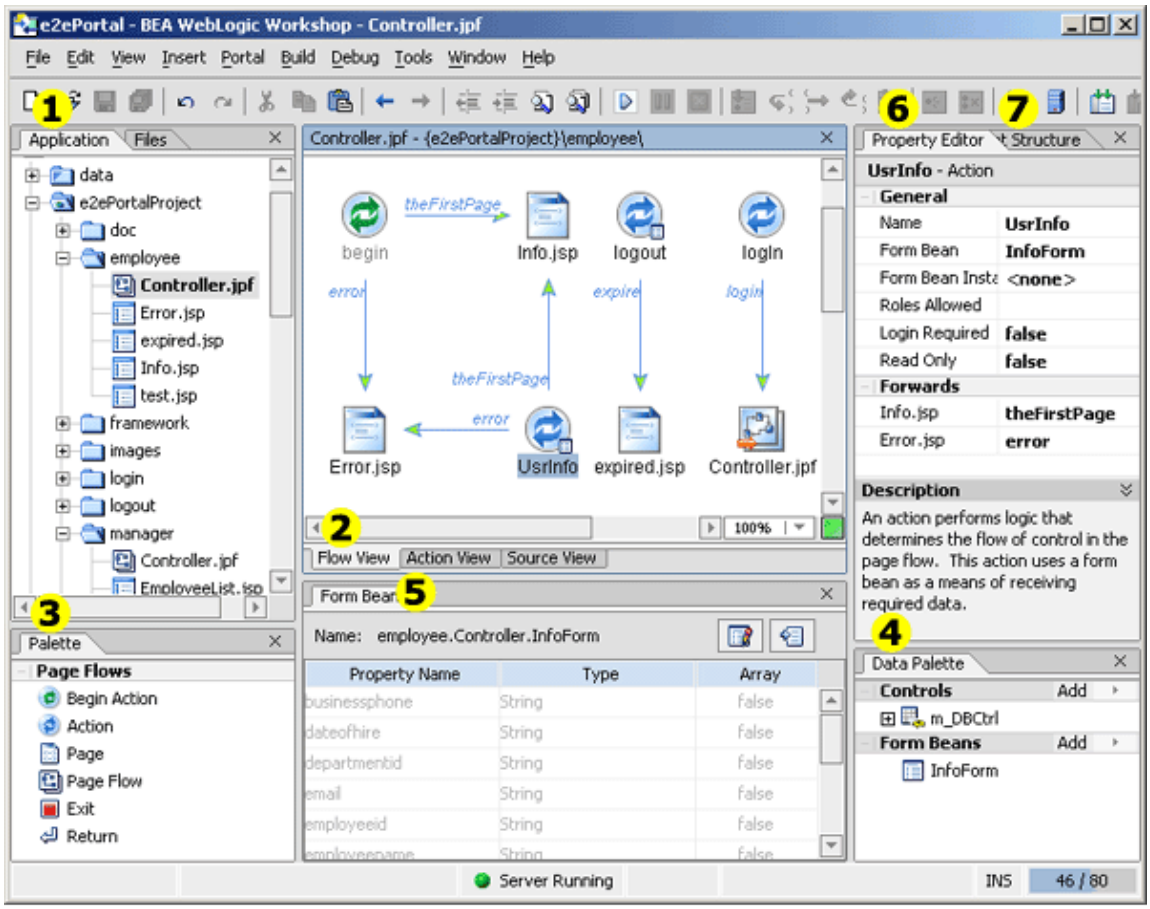
Within a Web application, a *page flow* links together multiple Web pages in a particular sequence, thus determining the user's path through those pages and the associated data. A page flow consists of a Java class with specially designed annotations, methods, and forms that control the behavior of Web application components; it is based on the Apache Struts framework.

You can include one or more JSP files in the page flow by adding them to the page flow folder. The JSP files can use special tags, which are understood by the page flow runtime, to raise actions. The actions in the JSP correspond to action methods that are defined in the page flow. Actions can be used to perform site navigation, pass data, or invoke back−end business logic via controls. The business logic in the page flow class is separate from the presentation code defined in the JSP files.

WebLogic Workshop makes it easy to create a page flow: click the target folder and choose File > New > Page Flow. The Page Flow Wizard steps you through the process of creating an initial page flow and opens it in the Page Flow Designer.

The Page Flow Designer is a graphic tool that enables you to design page flows such as the Employee page flow shown in the following figure.

Page Flow Designer Flow View



The following table describes the page flow design tools shown in the previous figure.

Tools for Designing Page Flows

Callout #	Use this tool...	For these tasks...
1		

Application Window

Create, view, and edit page flow files in your portal application projects. The names of page flow files end in .jpf.

Each page flow consists of a folder that contains a single controller file, any JSP files that are referenced, and any other application files required by the page flow. JSP files to be included in a page flow must reside within the page flow folder.

2

Flow, Action, and Source Views

Design your page flow in this area. Switch between the following views, as required:

- Flow View Build a graphical relationship between the pages in the Web application and the actions that link the pages.
- Action View Define the actions raised by the action icons in your page flow (see Figure: Page Flow Designer Action View).
- Source View Edit the source code directly to add business logic.

Changes made in one view are automatically reflected in the other.

3

Palette Window

Create a graph of icons in your page flow by dragging components from the Palette window and dropping them onto the Flow View canvas. Icon types include:

- Begin Action begin() method that defines the page that is loaded and the actions that are executed when the page flow is started. This icon is required and added automatically for you when you create a page flow.
- Action Method for an action that you define in the page flow controller class.
- Page JSP page that is loaded when an action method runs.
- Page Flow Page flow that is loaded when an action method runs.
- Exit done() method that defines what happens when the user exits the current page flow.
- Return to Action Redirect method that returns control to a specific action.
- Return to Page Redirect method that returns control to a specific page.

4

Data Palette Window

Define instances of Java controls and form beans to use in the page flow.

Java controls make it easy to access enterprise resources, such as Enterprise Java Beans (EJBs) and databases, from within your application. A control handles the work of connecting to the enterprise resource for you, so that you can focus on the business logic of your application.

Form beans define variables that are used to store form data. If an action defines a form bean, a small box appears in the lower right−hand corner of the action icon. For example, the previous figure shows that form beans are defined for the UsrInfo and logout methods.

5

Form Bean Window

Edit a form bean defined for the selected action. If no form beans are defined for the selected action, the window is empty.

6

Property Editor Window

Set properties for the component that is currently selected in the page flow. You can select a page flow icon by clicking on it in the Design View canvas or by selecting its name in the Document Structure window (described below).

7

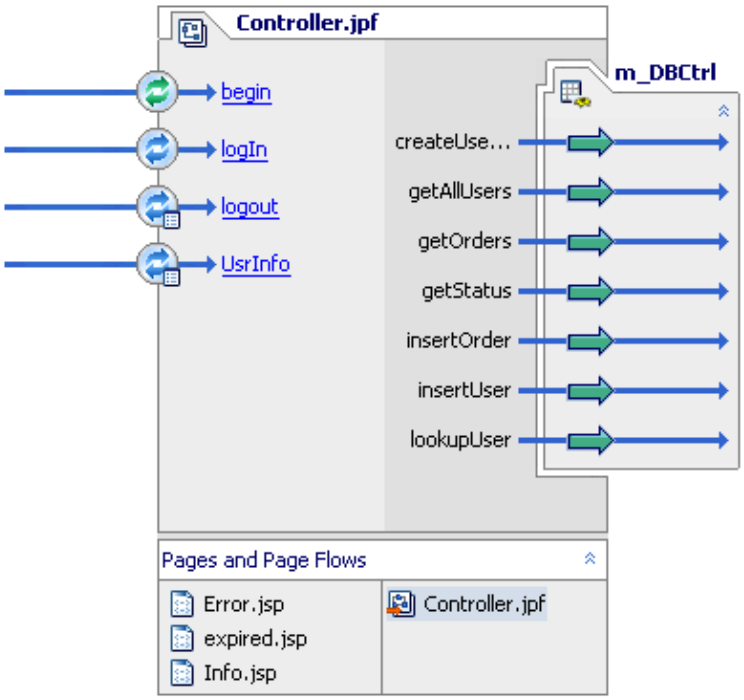
Document Structure Window

View a summary of the Java entities and page flow icons that comprise the page flow (see Figure: Document Structure Window for a Page Flow). The summary includes the Java classes, methods and signatures, variables, and inner classes defined for the page flow.

You can select a Java entity or page flow icon in the Document Structure window by clicking on it. When you select a page flow icon, it is selected in the Flow or Action View canvas and you can edit its properties in the Property Editor window. You can double−click on a Java entity or page flow icon to jump to its code location in Source View.

The following figure shows the Action View for the Employee page flow.

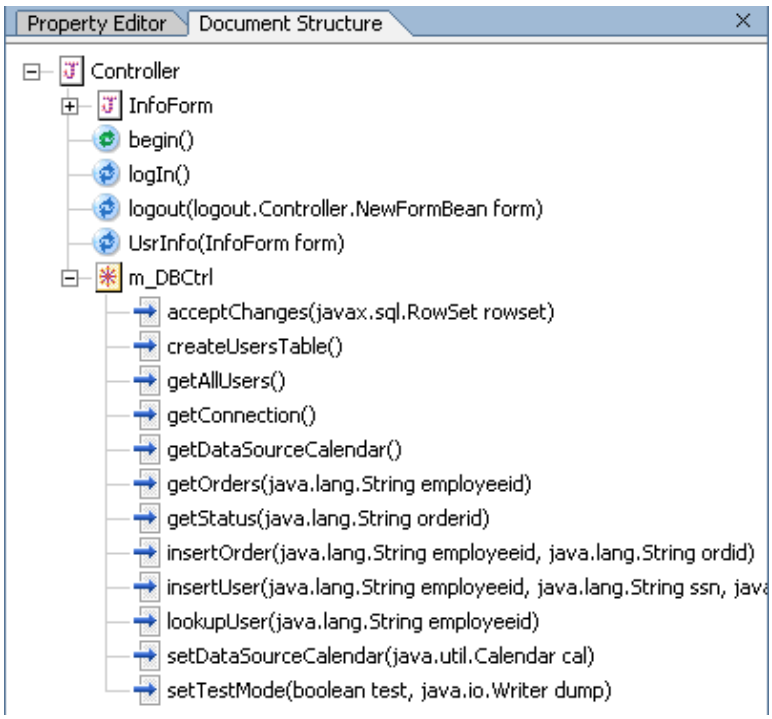
Page Flow Designer Action View



The Action View displays all actions and Java controls defined for the page flow, and the pages and page flows referenced.

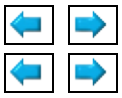
The following figure shows the Document Structure window for a page flow.

Document Structure Window for a Page Flow



To learn more about designing page flows:

- View a demo, Designing a Page Flow, that shows how to use the page flow design tools.
- See Guide to Building Page Flows in the *WebLogic Workshop Help*.

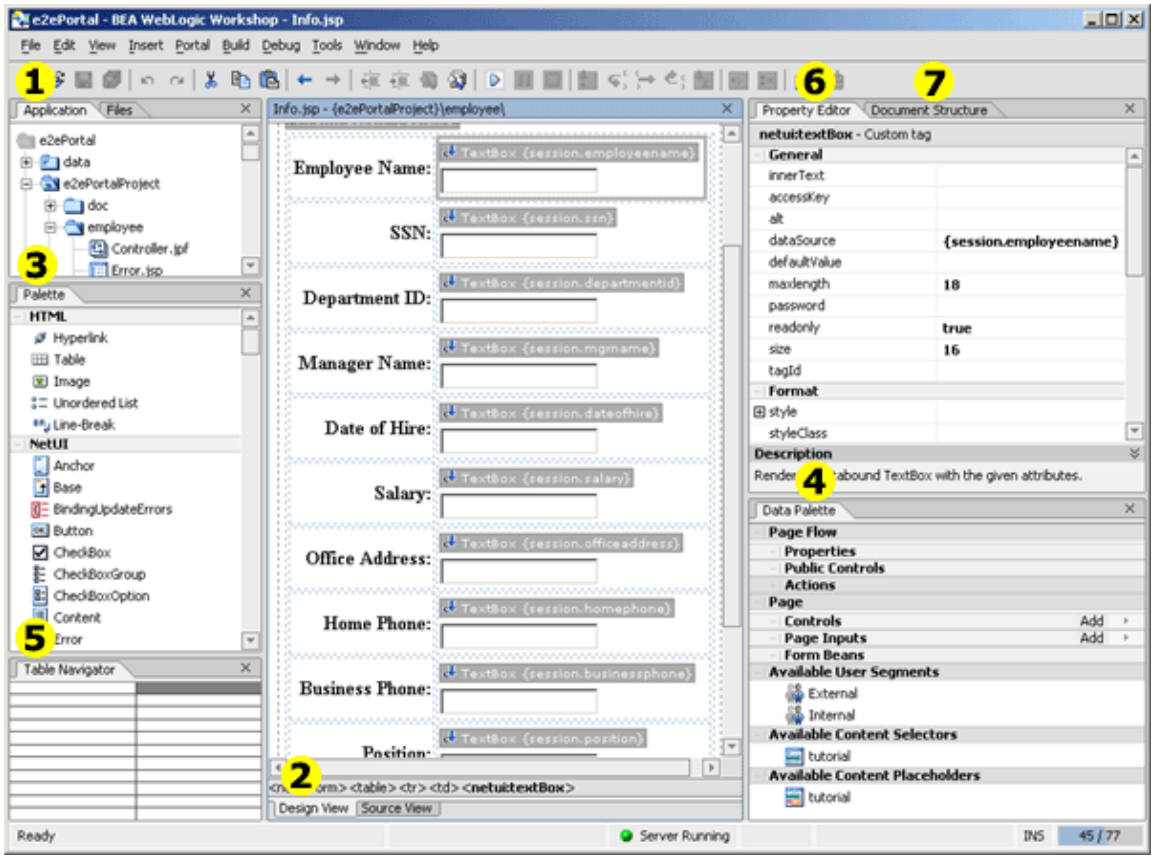


Creating JSP Content

JSPs provide a convenient way to present dynamically generated content over the Internet. JSPs are used to implement portlets and can contain static HTML code, JSP tag libraries, JSP scriptlets that access EJB components, or any other application functionality available with the application server.

WebLogic Workshop makes it easy to create a JSP: click the target folder and choose File > New > JSP File. The JSP Editor opens and displays a JSP file with basic HTML content.

JSP Editor



The following table describes the JSP editing tools shown in the previous figure.

Tools for Designing JSPs

Callout #	Use this tool...	For these tasks...
1		

Application Window

Create, view, and edit JSP files in your portal application projects. The names of JSP files end in .jsp.

2

Design and Source Views

Design your JSP in this area. Switch between the Design and Source code views, as required; changes made in one view are automatically reflected in the other.

In Design View, the HTML elements on the JSP page are rendered as they would be shown in a browser. The Java code and other elements are rendered in a schematic fashion.

3

Palette Window

Add JSP tags to your JSP content by dragging components from the Palette window and dropping them onto the Design View canvas.

JSP tags can be categorized as follows:

- HTML Standard HTML element tags.
- NetUI Page flow tags, including standard, data binding, and template tags.
- Portal WebLogic Portal tags for content rendering, personalization, skeleton rendering, and so on.
- Custom controls Tags for custom controls that you define for the project. The steps required to build a custom control are described in Building a Custom Control.
- Client−specific content Tags for client−specific content.

WebLogic Workshop includes the required tag libraries in the WEB−INF folder. The *.tld files are J2EE−standard tag library descriptor files. The *.tldx files are WebLogic Workshop specific files that contain information about how the tags behave within the IDE.

4

Data Palette Window

View or add components in the following categories:

- Page Add and view controls and view defined form beans.
- Page Flow View the properties, controls, and actions defined for the page flow.

5

Table Navigator Window

Navigate through the rows and columns of the selected table, and add, merge, or delete select cells and rows.

6

Property Editor Window

Set properties for the JSP file component that is currently selected. You can select a JSP tag component by clicking on it in the Design View canvas, n the hierarchical tag structure shown in the Design View status bar (see Figure: JSP Tag Hierarchy in Design View Status Bar), or in the Document Structure window (described below).

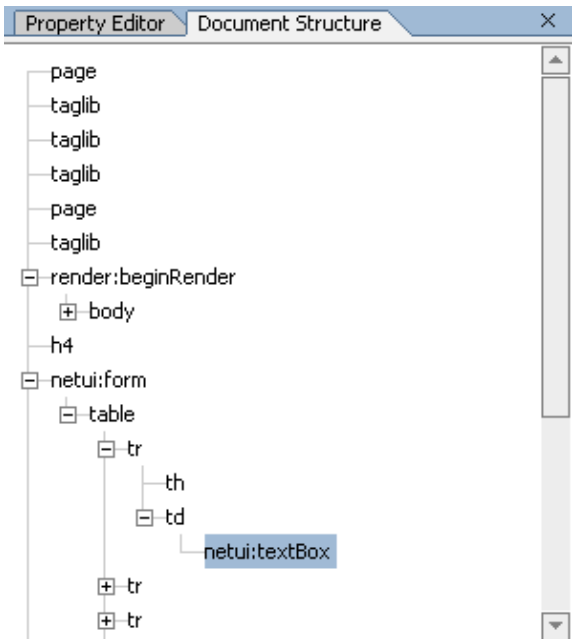
7

Document Structure Window

View the JSP tags in the JSP file in a hierarchical structure (see Figure: Document Structure Window for a JSP File). You can click any JSP component in the Document Structure window to select it in the Design View and edit its properties.

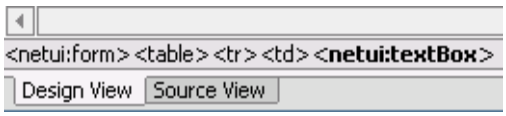
The following figure shows the Document Structure window for a JSP file.

Document Structure Window for a JSP File



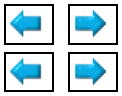
The current JSP tag hierarchy is also reflected in the Design View status bar, as shown in the following figure.

JSP Tag Hierarchy in Design View Status Bar



To learn more about creating JSP content:

- View a demo, Creating JSP Content, that shows how to use the JSP design tools.
- See Designing User Interfaces in JSPs in the *WebLogic Workshop Help*.



Administering Portals With WebLogic Portal

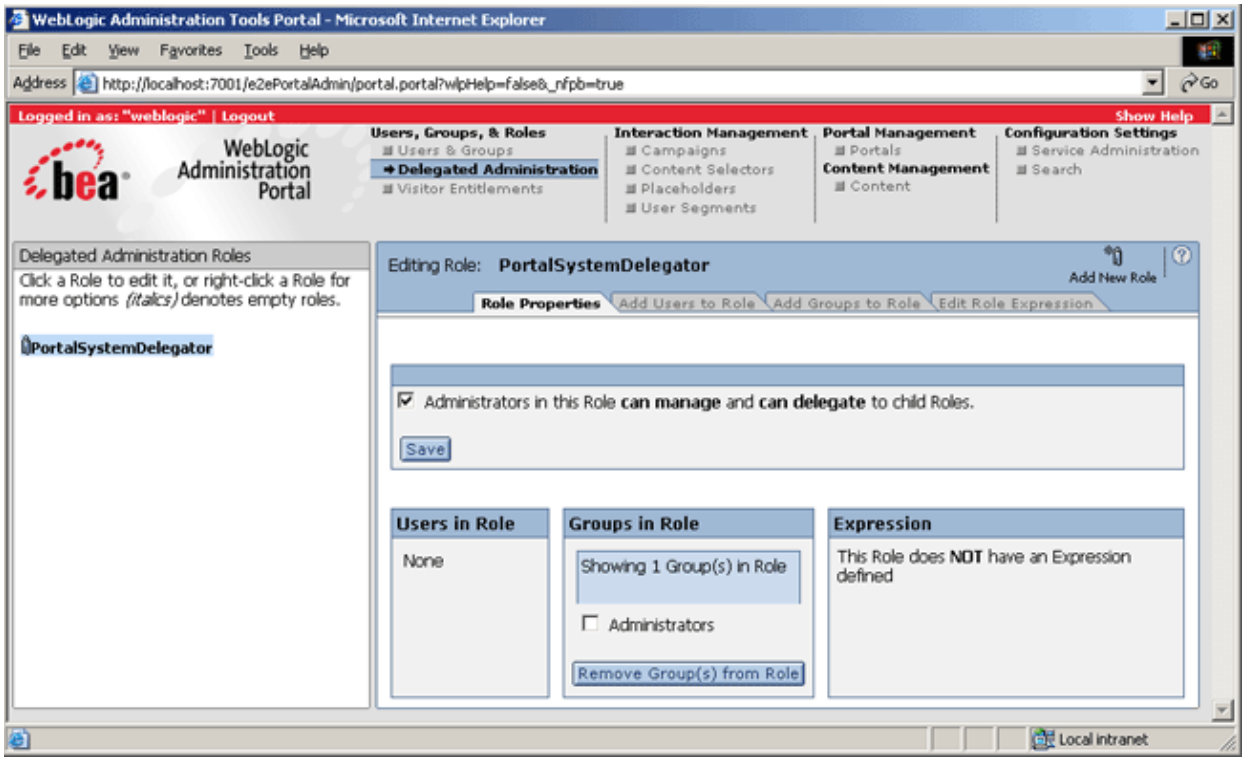
WebLogic Portal provides a control structure that supports the full portal lifecycle, including design, development, deployment, and management. Once the portal designer has developed the resources required for a portal Web site, the portal administrator can use those resources to assemble, maintain, and modify multiple portals.

Portal administration involves many traditional system administration activities, as well as business user tasks that control the behavior, content, and appearance of portals.

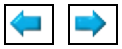
The WebLogic Administration Portal, shown in the following figure, enables portal administrators to manage the following portal features:

- Users, Groups, and Roles Set visitor and administrative users, group memberships, and global roles.
- Interaction Management Manage campaigns, content selectors, placeholders, and user segments.
- Portal Management Manage portals, desktops, books, pages, portlets, and other portal resources.
- Content Management Manage repositories, types, and content.
- Configuration Settings Manage portal server and search configuration settings.

WebLogic Administration Portal



To learn more about managing portals, see the WebLogic Administration Portal Help, available on E−docs.



Reviewing the Avitek Intranet Portal

The Avitek intranet portal enables employees and managers to log in to the corporate intranet. When you log in, you access a customized version of the intranet portal for either an employee or a manager.

The following table defines the user names defined for the WebLogic Platform Tour.

User Names Defined for the WebLogic Platform Tour

This user name... And password...	Enables...
john employee	John Smith, an employee of Avitek, to: • View his employee profile • Submit orders for office equipment and check the status of those orders
rachel emanager	Rachel Burns, a manager at Avitek, to: • View a list of her direct employees • Approve or reject orders for office equipment issued by her direct employees and check the status of those orders

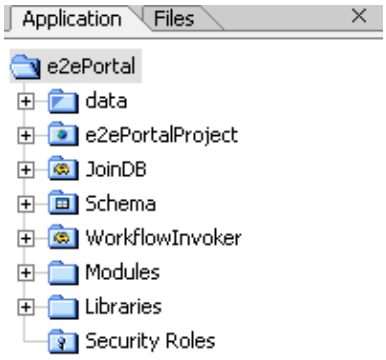
The e2ePortal Web application defines the Avitek intranet portal. To view the e2ePortal Web application in WebLogic Workshop:

1. Open the e2ePortal application, as follows:
    a. Choose **File** > **Open** > **Application**.
    b. Navigate to the samples\platform\end2end\e2ePortal folder of the WebLogic Platform installation.
    c. Select **e2ePortal.work**.

       The names of application files end in .work.

    d. Click **Open** to open the application.
2. If the application files are not currently displayed in WebLogic Workshop, choose **View** > **Application**.

   The e2ePortal application information is displayed, as follows:



The following table describes each of the folders in the e2ePortal application.

e2ePortal Application Folders

Folder	Contents
data	Data files required to support WebLogic Portal features, such as personalization, campaigns, and so on. This folder is standard in all application projects. You can modify the default contents based on your application requirements.
e2ePortalProject	Application project files, including portal, portlet, page flow, and JSP content files for the sample portals. This folder also contains the following standard application subdirectories: • resources Standard Web application folder containing Portal application resources such as sample CSS, images, and template JSPs. • WEB−INF Standard Web application folder containing deployment descriptors and validation files. The contents of this folder are described in the sections that follows.
Schema	Schema definitions.
JoinDB	Custom Java control that coordinates access to multiple databases. The contents of this folder are described in detail in Reviewing the Custom Controls for the Avitek Intranet Portal.
WorkflowInvoker	Custom Java control that invokes a business process from the portal. The contents of this folder are described in detail in Reviewing the Custom Controls for the Avitek Intranet Portal.
Modules	

Application and portal system JAR files. This folder is standard in all application projects. You can modify the default contents based on your application requirements.

Libraries

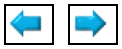
Run−time related library files. This folder is standard in all application projects. You can modify the default contents based on your application requirements.

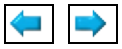
Security Roles

Security role information (this folder is empty for the e2ePortal Web application). This folder is standard in all application projects. You can modify the default contents based on your application requirements.

The following sections describe the contents of the e2ePortalProject folder, including the portals, portlets, page flows, and JSP content files that make up the Avitek intranet portal.

Before proceeding, open WebLogic Workshop and expand the e2ePortalProject folder in the Application window. To view the contents of any portal or portlet, double−click its name in the Application window.



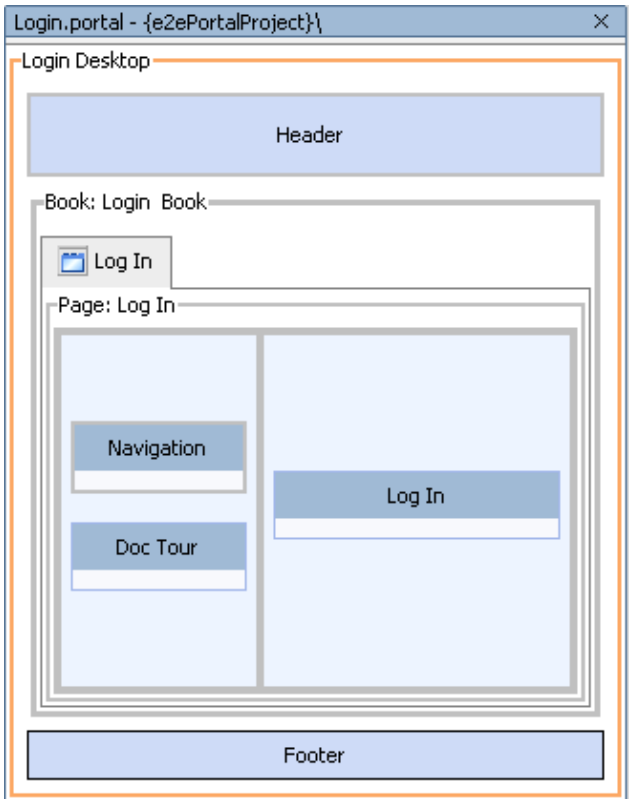


Reviewing the Portals

The e2ePortalProject Web application consists of three portals: Log In, Employee, and Manager.

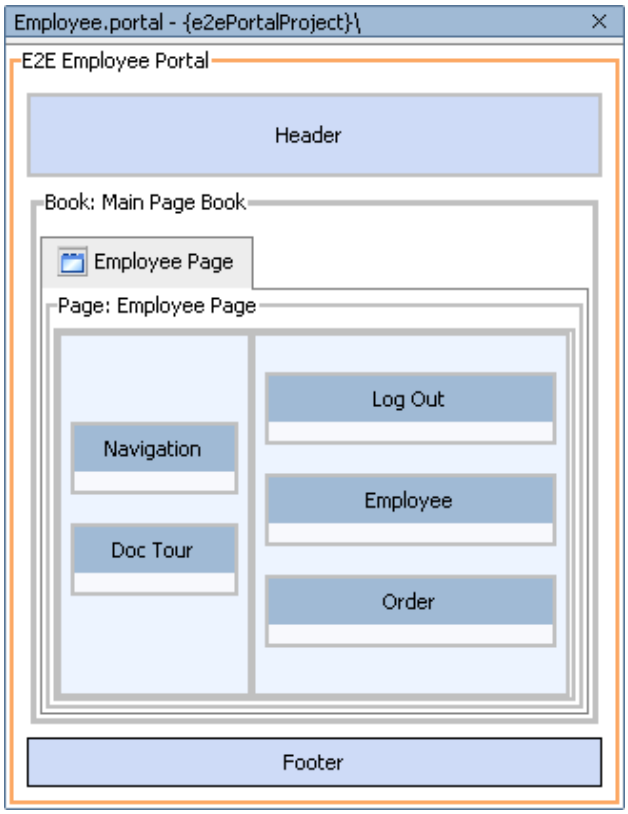
- **Log In** Enables employees to log in to and interface with the Employee Management System. The source file for this portal is Login.portal.

Log In Portal



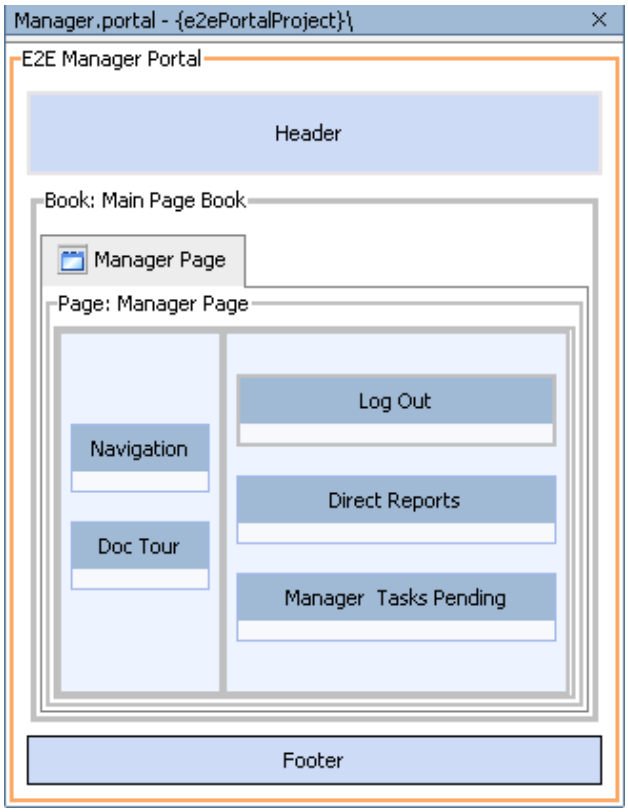
- **Employee** Enables employees to view their employee profiles and order office equipment. The source file for this portal is Employee.portal.

Employee Portal



- **Manager** Enables managers to view a list of their direct employees and to approve office equipment orders. The source file for this portal is Manager.portal.

Manager Portal



As shown, each portal in the e2ePortalProject application project defines a hierarchical structure with the following elements:

- Header and Footer
- Body containing:
    - ♦ Single book and page
    - ♦ Two−column layout
    - ♦ Placeholders for each portlet
    - ♦ Set of portlets (described in Reviewing the Portlets)

The framework folder defines the look and feel elements available to the application project. The Look and Feel property in the Property Editor window is set to default for each of the portal files.

The portlets are described in detail in the next section.

Reviewing the Portlets

The following table defines the portlets that are defined for each of the three portals.

Portlets for the Employee Management System

Portlet

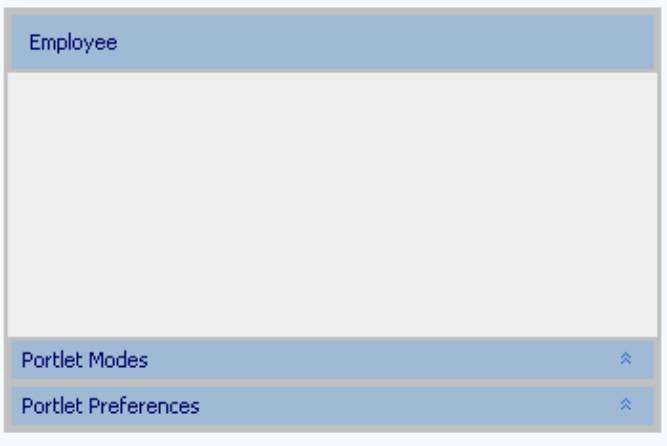
Parent Portal	Description
Direct Reports	
Manager	Accesses the Employee Information database to display a list of employees who report to a particular manager.
Doc Tour	
All Portals	Provides a context−sensitive link to the *WebLogic Platform Tour Guide* (this guide) for more detailed information about stepping through the tour.
Employee	
Employee	Accesses the Employee Information database to display an employee profile.
Log In	
Log In	Enables users to log in to the Avitek intranet portal, working with the WebLogic Server security system to provide secure access to the portal.
Log Out	
All portals (except Log In)	Enables users to log out of the portal.
Manager Tasks Pending	
Manager	Communicates with the Office Equipment Order Management system to allow managers to view office equipment orders issued by their direct employees, and to approve or reject pending orders.
Navigation	
All Portals	Displays context−sensitive information about the currently displayed portal and portlets and provides a context−sensitive link to the *WebLogic Platform Tour Guide* (this guide) for more detailed information.
Order	

Employee

Communicates with the Office Equipment Order Management system to allow employees to submit orders for office equipment or to view pending orders.

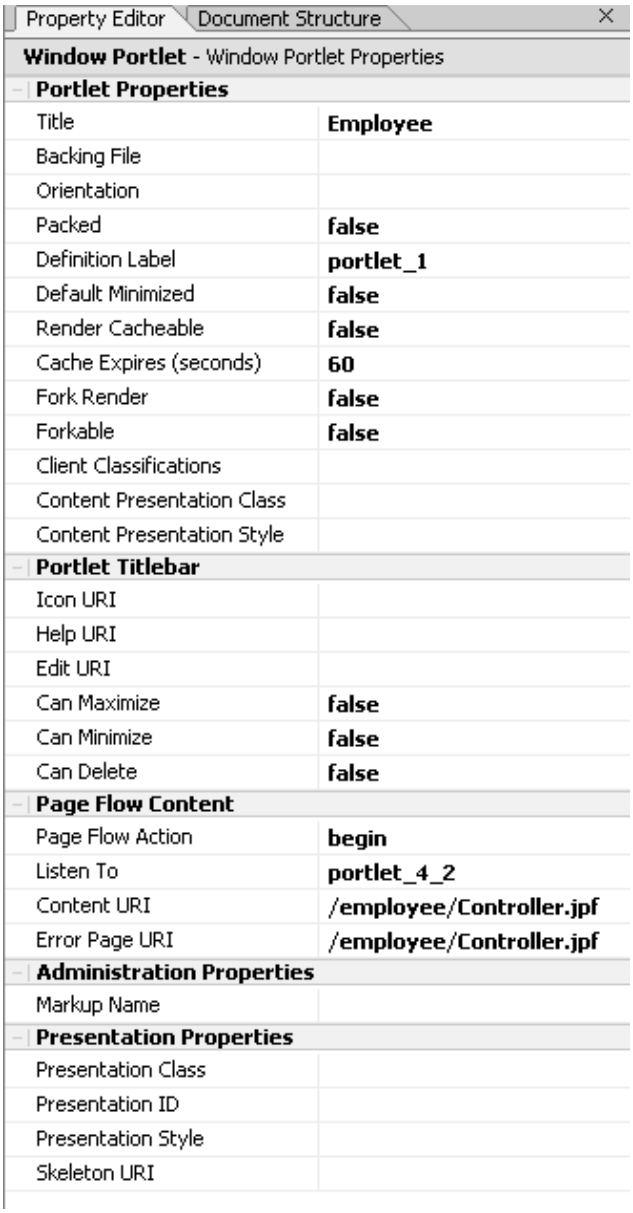
For example, the following figure shows the Employee portlet in Design View.

Employee Portlet in Design View



The following figure shows an example of the Property Editor window settings for the Employee portlet.

Portlet Property Settings for the Employee Portlet

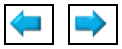


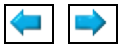
The majority of the property settings shown in the previous figure are the same across all portlets in the Avitek intranet portal. The Content and Portlet Title property settings are unique to each portlet.

The following sections describe the following portlets:

- Direct Reports Portlet
- Doc Tour Portlet
- Employee Portlet
- Log In Portlet
- Log Out Portlet
- Manager Tasks Pending Portlet
- Navigation Portlet
- Order Portlet

To display any portlet in Design View, double−click the name of the portlet in the Application Window.



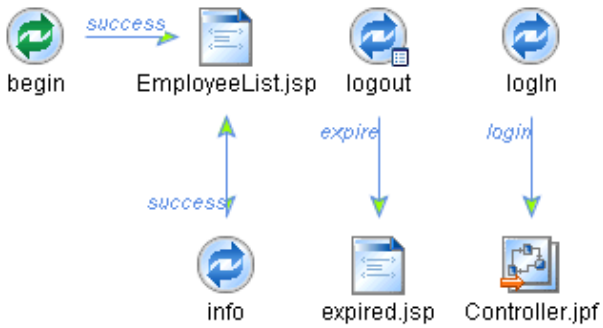


Direct Reports Portlet

The Direct Reports portlet (Manager.portlet) accesses the Employee Information database to display a list of the employees who report directly to a particular manager. The Content URI property in the Property Editor window specifies the following file as the initial page flow file referenced by the Direct Reports portlet: /manager/Controller.jpf.

The following figure shows the controller file, Controller.jpf, for the Direct Reports page flow in the Flow View canvas of the Page Flow Designer.

Controller File for the Direct Reports Page Flow



The following table describes each component of the controller file for the Direct Reports page flow.

Components of the Controller File for the Direct Reports Page Flow

Component	Function
begin Icon	Defines the employeeList() method that retrieves a list of the current manager's direct employees and passes control to EmployeeList.jsp. The employeeList() method uses an instance of the UsersDBControl database control, m_DBCtrl, to retrieve employee information from the Employee Information database. The UsersDBControl database control is described in detail in Viewing Employee Information.
EmployeeList.jsp	Displays a list of the current manager's direct employees. First, it uses a netui−data:callPageFlow JSP page flow tag to call the employeeList() method defined in the controller file. Then, it uses the netui−data:repeater, netui−data:repeaterHeader, netui−data:repeaterItem, and netui:label JSP page flow tags to render the contents. For example: `<netui:form action="info">`

```
<netui-data:callPageFlow method="employeeList" resultId="array"/>
<table border="0">
<netui-data:repeater dataSource="{pageContext.array}">
   <netui-data:repeaterHeader>
   <table border="0.5">
   <tr>
   <td><b>Employee Name</b></td>
   <td></td>
   <td><b>Employee Email</b></td>
   </tr>
</netui-data:repeaterHeader>
<netui-data:repeaterItem>
   <tr>
     <td>
     <netui:label value="{container.item.employeename}"/>
     </td>
     <td>
     <netui:label value="{container.item.email}"/>
     </td>
   </tr>
</netui-data:repeaterItem>
</netui-data:repeater>
</table>
</netui:form>
```

To learn more about the JSP page flow tags listed here, see Designing User Interfaces in JSPs in the *WebLogic Workshop Help*.

info Action

Dummy action for form defined in EmployeeList.jsp. Passes control to EmployeeList.jsp.

logout Action

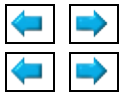
Passes control to the expired.jsp file. This action is triggered, when the user logs off the system, by the Log Out page flow controller file, described in Log Out Portlet.

expired.jsp

Displays a Data Expired message when the user logs off the system to indicate data within the current session has expired.

logIn Action

Passes control to the begin action of the Controller.jpf page flow. This action is triggered, when the user selects **Log In Again** from the Log Out portlet.

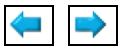


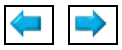
Doc Tour Portlet

The Doc Tour portlet (Doc.portlet) provides a context−sensitive link to the *WebLogic Platform Tour Guide* (this guide) for detailed information about stepping through the tour. The Content URI property in the Property Editor window specifies the following file as the page flow file referenced by the Doc Tour portlet as

follows: /doc/Controller.jpf.

To provide context−sensitive information, the Controller.jpf file calls the appropriate JSP files based on the user actions performed within the other portlets.



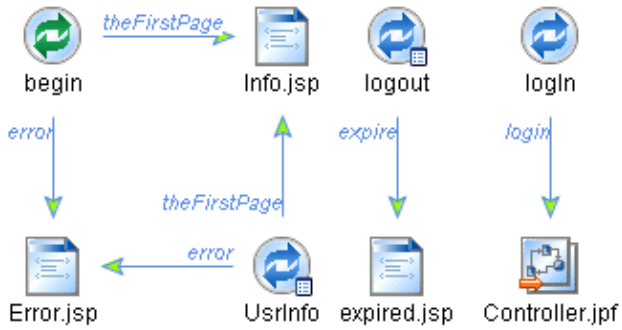


Employee Portlet

The Employee portlet (Employee.portlet) accesses the Employee Information database to display an employee profile. The Content URI property in the Property Editor window specifies the following file as the page flow file referenced by the Employee portlet as follows: /employee/Controller.jpf.

The following figure shows the Employee page flow controller file, Controller.jpf, in the Flow View canvas of the Page Flow Designer.

Controller File for Employee Page Flow



The following table describes each component of the controller file for the Employee page flow.

Components of the Controller File for the Employee Page Flow

Component	Function
begin Icon	Retrieves the user name for the current user and passes control to Info.jsp (or to Error.jsp in the event of an error).

The begin() method uses an instance of the UsersDBControl database control, m_DBCtrl, to retrieve the user name using the employee ID. For example:

```
joindb.UsersDBControl.User user= null;
.
.
.
String empId=SubjectUtils.getUsername(sub);
user = m_DBCtrl.lookupUser(empId);
```

The UsersDBControl database control is described in detail in Viewing Employee Information.

Info.jsp

Displays the employee profile for the current user. Defines an input form to collect the information about the employee using the netui:form JSP page flow tag. It associates the contents of the form with the form bean defined by the UsrInfo action. For example:

```
<netui:form action="/UsrInfo" focus="employeename">
```

To learn more about the netui:form page flow tag, see Designing User Interfaces in JSPs in the *WebLogic Workshop Help*.

Error.jsp

Displays an error message in the event of an error.

UsrInfo Action

Retrieves the employee information for the current user using a form bean with properties that correspond to the data fields in the input form. It then passes control to Info.jsp (or to Error.jsp in the event of an error).

The UsrInfo action uses an instance of the UsersDBControl database control, m_DBCtrl, to retrieve employee information from the Employee Information database. The UsersDBControl database control is described in detail in Viewing Employee Information.

logout Action

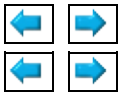
Passes control to the expired.jsp file. This action is triggered, when the user logs off the system, by the Log Out page flow controller file, described in Log Out Portlet.

expired.jsp

Displays a Data Expired message when the user logs off the system to indicate data within the current session has expired.

logIn Action

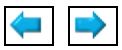
Passes control to the begin action of the Controller.jpf page flow. This action is triggered, when the user selects **Log In Again** from the Log Out portlet.

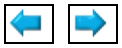


Log In Portlet

The Log In portlet (Login.portlet) enables users to log in to the Avitek intranet portal, working with the WebLogic Server security system to provide secure access to the Avitek intranet portal. The Content URI property in the Property Editor window specifies the following file as the initial JSP file displayed in the Log In portlet as follows: /login/login.jsp

The secure log in process is described in detail in Configuring Security in Web Applications.



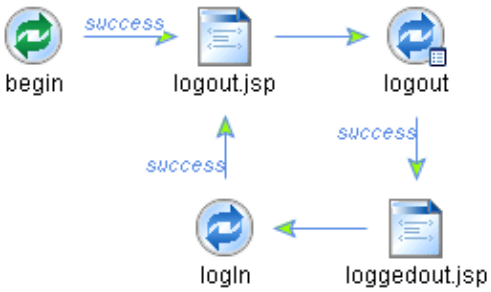


Log Out Portlet

The Log Out portlet (Logout.portlet) enables a user to log out of the Avitek intranet portal. The Content URI property in the Property Editor window specifies the following file as the page flow file referenced by the Log Out portlet: /logout/Controller.jpf.

The following figure shows the controller file, Controller.jpf, for the Log Out page flow in the Flow View canvas of the Page Flow Designer.

Controller File for the Log Out Page Flow



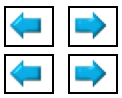
The following table describes each component of the controller file for the Log Out page flow.

Components of the Controller File for the Log Out Page Flow

Component	Function
begin Icon	Passes control to the logout.jsp file.
logout.jsp	Displays the name of the current user and enables a user to log out.
logout Action	Initiates the logout process when a user clicks the Log Out button. Triggers the logout actions defined in the Employee and Order portlets to expire any current data.
loggedout.jsp	Displays a message indicating that the user has been logged out.

logIn Action

Returns the user to the login/logout.jsp file.



Manager Tasks Pending Portlet

The Manager Tasks Pending portlet (Worklist.portlet) communicates with the Office Equipment Order Management system to display office equipment orders issued by a manager's direct employees, and to enable the manager to approve or reject pending orders. The Content URI property in the Property Editor window specifies the following file as the page flow file referenced by the Manager Tasks Pending portlet: /worklist/Controller.jpf.

The following figure shows the controller file, Controller.jpf, for the Manager Tasks Pending page flow in the Flow View canvas of the Page Flow Designer.

Controller File for the Manager Tasks Pending Page Flow

The following table describes each component of the controller file for the Manager Tasks Pending page flow.

Components of the Controller File for the Manager Tasks Pending Page Flow

Component	Function
begin Icon	Defines a Worklist TaskSelector to retrieve a list of tasks assigned to the current user. It then passes control to the worklist.jsp file. The begin() method uses an instance of the ManagerWorker task worker control, worker, to communicate with the manager's Worklist user interface.

A *Worklist user interface* allows individuals to interact with a running business process to handle tasks assigned to them. A *task worker control* provides an interface to a Worklist user interface, allowing your business process to operate on a current task.

The worker task control is used to retrieve the current tasks assigned. For example:

```
private worklist.ManagerWorker worker;
/**
  * @jpf:action
  * @jpf:forward name="success" path="worklist.jsp"
*/
protected Forward begin()
{
   Subject sub= Security.getCurrentSubject();
   // by default display the owned tasks
   TaskSelector selector = new TaskSelector();
.
.
.
   TaskInfo[] infos = worker.getTaskInfos(selector);
.
.
.
}
```

worklist.jsp

Provides a user interface to the Worklist, displaying information about all pending tasks assigned to the current user, including task name, task state, and associated employee name. It uses the netui−data:repeater, netui−data:repeaterHeader, netui−data:repeaterItem, and netui:label JSP page flow tags to render the contents.

The worklist.jsp files provide links to raise an action using the netui−anchor JSP page flow tag. For example:

```
<netui:anchor action="taskDetail">Respond to the Task
<netui:parameter name="taskId"
   value="{container.item.taskId}"/>
</netui:anchor>
```

To learn more about the JSP page flow tags listed here, see Designing User Interfaces in JSPs in the *WebLogic Workshop Help*.

taskDetail Action

Gets details about a selected task using worker, an instance of the ManagerWorker task worker control. For example:

```
TaskInfo[] infos = worker.getTaskInfo(taskId);
```

It then passes control to the response.jsp file. This action is triggered, when the user clicks Respond to Task.

response.jsp

Displays the Task Response text box, prompting the manager to respond to the pending task.

setReplyXML Action

Retrieves the manager's response using a form bean that defines the responseXML property that corresponds to the data field in the input form. It then passes control to the worklist.jsp file (or to Error.jsp in the event of an error).

The setReply action uses an instance of worker, an instance of the ManagerWorker task worker control to set the task response. For example:

```
worker.setTaskResponseString(response, taskInfo.getTaskId());
worker.startTask(taskInfo.getTaskId());
worker.completeTask(taskInfo.getTaskId());
```

error.jsp

Displays an error message in the event of an error.

logout Action

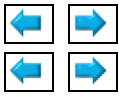
Passes control to the expired.jsp file. This action is triggered, when the user logs off the system, by the Log Out page flow controller file, described in Log Out Portlet.

expired.jsp

Displays a Data Expired message when the user logs off the system to indicate data within the current session has expired.

logIn Action

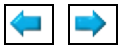
Passes control to the worklist.jsp file. This action is triggered, when the user selects **Log In Again** from the Log Out portlet.

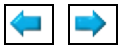


Navigation Portlet

The Navigation portlet (Navigation.portlet) displays context−sensitive information about the currently displayed portal and portlets. The Content URI property in the Property Editor window specifies the following file as the page flow file referenced by the Navigation portlet as follows: /navigation/Controller.jpf.

To provide context−sensitive information, the Controller.jpf file calls the appropriate JSP files based on the user actions performed within the other portlets.





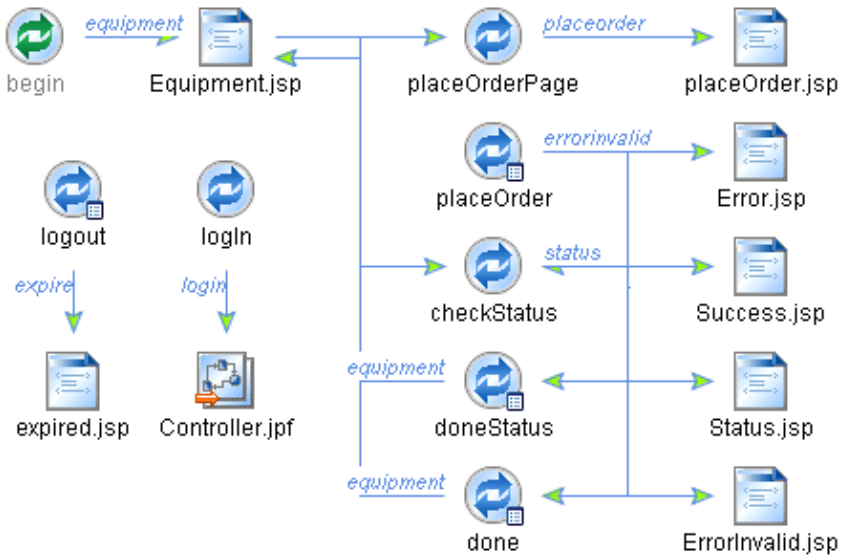
Order Portlet

The Order portlet (Order.portlet) communicates with the Office Equipment Order Management system to enable employees to submit new office equipment orders or to view pending orders. The Content URI property in the Property Editor window specifies the following file as the initial page flow file referenced by

the Order portlet: /order/Controller.jpf.

The following figure shows the Order page flow controller file, Controller.jpf, in the Flow View canvas of the Page Flow Designer.

Controller File for the Order Page Flow



The following table describes each component in the controller file for the Order page flow.

Components of the Controller File for the Order Page Flow

Component	Function
begin Icon	Passes control to the Equipment.jsp file.

Equipment.jsp	Provides a user interface to the Office Equipment Order Management system, enabling users to place an order or view the status of pending orders. It provides links to raise actions using the netui−anchor JSP page flow tag, including placeOrderStatus and checkStatus. For example:
	
	`<netui:anchor action="placeOrderPage">`
	`  Place a New Order</netui:anchor></p>`
	`<netui:anchor action="checkStatus">`
	`  Check Order Staus</netui:anchor></p>`

placeOrderPage Action	Passes control to the placeOrder.jsp file.

placeOrder.jsp

Prompts the user for the ID of the item being ordered. It defines an input form to collect the item ID using the netui:form JSP page flow tag. It associates the contents of the form with the form bean defined by the placeOrder action. For example:

<netui:form action="/placeOrder" focus="ordid">

To learn more about the netui:form page flow tag, see Designing User Interfaces in JSPs in the *WebLogic Workshop Help*.

placeOrder Action

Retrieves the office equipment order and passes control to Success.jsp (or to Error.jsp in the event of an error). The placeOrder action defines a form bean with properties that correspond to the data fields in the order input form.

The placeOrder action uses m_WorkflowInvoker, an instance of the custom WorkflowInvoker control, to invoke the Order Requisition business process. The WorkflowInvoker custom control is described in detail in Building a Custom Control.

It also uses m_DBCtrl, an instance of the UsersDBControl database control, m_DBCtrl, to insert the order into the Employee Information database. The UsersDBControl database control is described in detail in Viewing Employee Information.

Success.jsp

Displays a message, indicating that the order was placed successfully, and prompts the user to check the status of all pending orders or click Done. It defines an input form using the netui:form JSP page flow tag and associates its contents with the form bean defined by the done action. For example:

<netui:form action="done">

If the user clicks Check Status, the checkStatus action is raised; if the user clicks Done, the done action is raised. For example:

```
<netui:button type="submit"
   value="Check Order Status" action="checkStatus"></netui:button>
<netui:imageButton value="Done"
src="/e2ePortalProject/framework/skins/default/images/done.gif">
</netui:imageButton>
```

Error.jsp

Displays an error message in the event of an error.

checkStatus Action

Retrieves the employee ID and uses the JoinDB custom control to coordinate database queries to the Employee Information and Office Equipment Order Management databases. For example:

```
m_Array= m_JoinDB.joinDB(m_employeeid);
```

It then passes control to the Status.jsp file.

The JoinDB custom control is described in detail in Building a Custom Control.

Status.jsp

Displays a list of the current manager's direct employees. First, it uses a netui−data:callPageFlow JSP page flow tag to call the checkStatusBusiness() method defined in the controller file. Then, it uses the netui−data:repeater, netui−data:repeaterHeader, netui−data:repeaterItem, and netui:label JSP page flow tags to render the contents. For example:

```
<netui−data:callPageFlow method="checkStatusBusiness" resultId="m_Array"/>
<netui−data:repeater dataSource="{pageContext.m_Array}">
<netui−data:repeaterHeader>
<table border="1">
<tr>
  <td><b>Order ID</b></td>
  <td><b>Status Description</b></td>
</tr>
</netui−data:repeaterHeader>
<netui−data:repeaterItem>
<tr>
  <td>
  <netui:label value="{container.item.m_ordId}" />
  </td>
  <td>
  <netui:label value="{container.item.m_statusDesc}" />
  </td>
</tr>
</netui−data:repeaterItem>
<netui−data:repeaterFooter>
</table>
</netui−data:repeaterFooter>
</netui−data:repeater>
```

To learn more about the JSP page flow tags listed here, see Designing User Interfaces in JSPs in the *WebLogic Workshop Help*.

DoneStatus Action

Returns control to the Equipment.jsp file.

Done Action

Returns control to the Equipment.jsp file.

logout Action

Passes control to the expired.jsp file. This action is triggered, when the user logs off the system, by the Log Out page flow controller file, described in Log Out Portlet.

expired.jsp

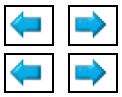
Displays a Data Expired message when the user logs off the system.

ErrorInvalid.jsp

Reprompts for information when user specifies an invalid entry.

logIn Action

Passes control to the begin action of the Controller.jpf page flow. This action is triggered, when the user selects **Log In Again** from the Log Out portlet.



Building a Custom Control

Java controls are reusable components you can use anywhere within a platform application. They provide a convenient way to incorporate access to resources and encapsulate business logic.

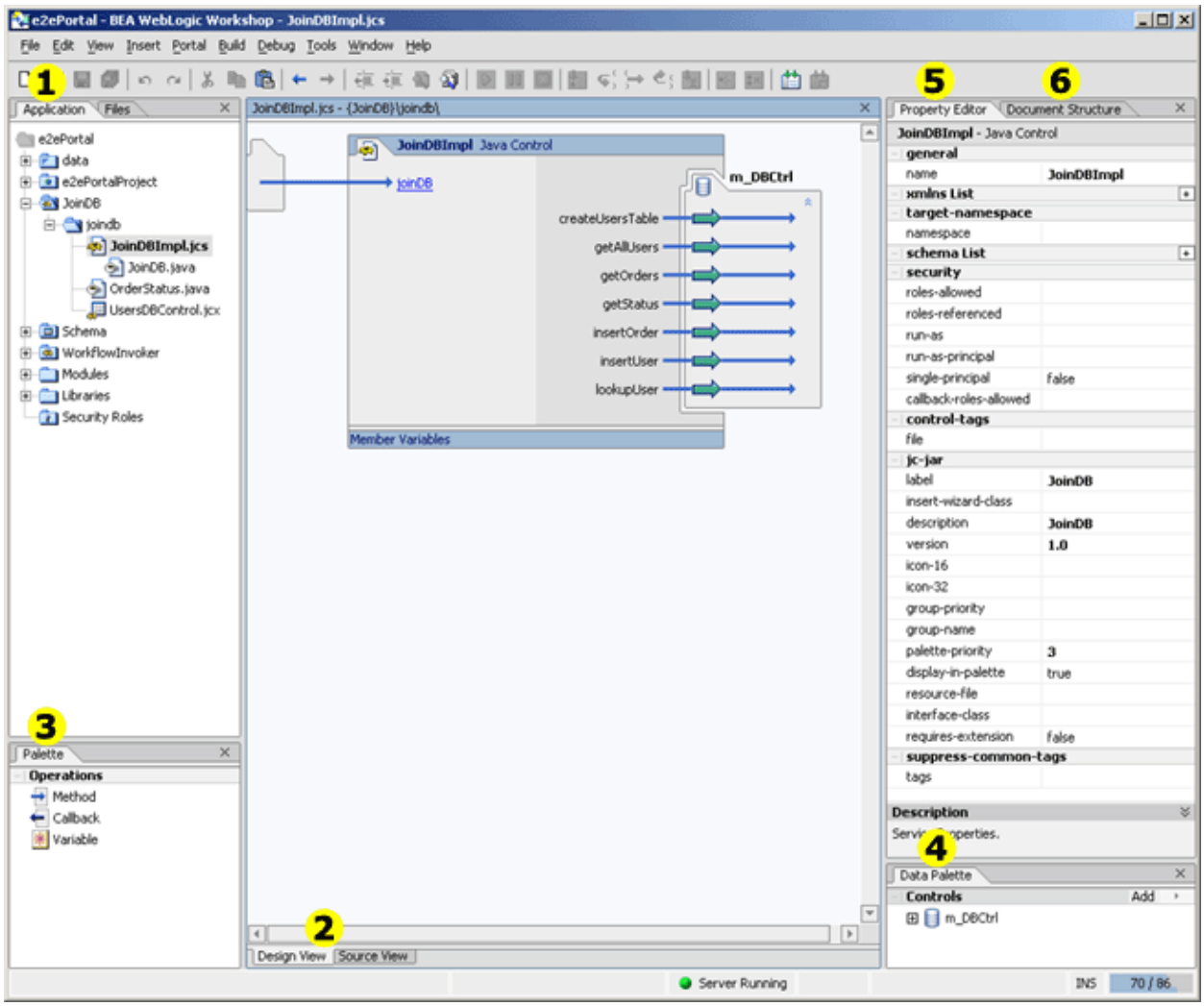
WebLogic Workshop includes a set of *built−in* Java controls, mostly designed to access resources, such as Enterprise Java Beans (EJBs) and databases, from within your application. For an example of how to use one, see Using a Built−In Control.

You can also create *custom* Java controls tailored to your project or application. Custom controls are based on the same framework that built−in controls are based. Unlike built−in controls, which give you access to an interface that extends the source, source files for custom controls are stored within the project and accessed directly from the locations in which they are stored. This arrangement enables you to separate and reuse the control code.

WebLogic Workshop makes it easy to create a custom control: click the target folder and choose File > New > Custom Java Control. The Custom Control Editor opens and displays a Java control source file. This file defines the implementation for your custom control. It can be identified by the string Impl that appears in its name, for example, JoinDBImpl.jcs. In addition, WebLogic Workshop automatically generates and maintains a Java class file that defines the public interface for the Java control you never need to edit this file.

The Custom Control Editor is a graphical tool that enables you to design a custom control and create methods to expose its functionality, such as the JoinDB custom control shown in the following figure.

Custom Control Editor



The following table describes the custom control design tools shown in the previous figure.

Tools for Designing Custom Control Files

Callout #	Use this tool...	For these tasks...
1	Application Window	Create, view, and edit custom control files in your portal application projects. The names of custom control files end in .jcs.
2	Design and Source Views	Design your custom control file in this area. Switch between the Design and Source code views, as required;

changes made in one view are automatically reflected in the other.

3

Palette Window

Add methods, callbacks, variables, and so on, by dragging components from the Palette window and dropping them onto the Design View canvas. The options available in the Palette window depend on the type of control you are building.

4

Data Palette Window

Define instances of Java controls for use in the custom control.

6

Property Editor Window

Set properties for the custom control component that is currently selected. You can select a custom control component by clicking on it in the Design View canvas or by selecting its name in the Document Structure window (described below).

7

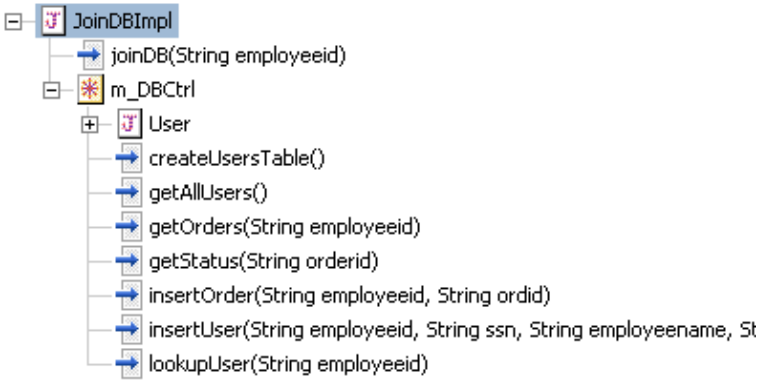
Document Structure Window

View a summary of the Java entities that comprise the control file (see Figure: Document Structure Window for a Custom Control). The summary includes the Java classes, methods and signatures, variables, and inner classes defined for the custom control.

You can select a Java entity in the Document Structure window by clicking on it. When you select a method, it is selected in the Design View canvas and you can edit its properties in the Property Editor window. You can double−click on a Java entity to jump to its code location in Source View.

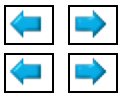
The following figure shows the Document Structure window for a custom control.

Document Structure Window for a Custom Control



To learn more about building custom controls:

- See Tutorial: Java Control in the *WebLogic Workshop Help*.
- See Building Custom Java Controls in the *WebLogic Workshop Help*.

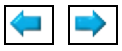


Reviewing the Custom Controls for the Avitek Intranet Portal

The e2ePortal application defines two custom controls:

- JoinDB Coordinates database queries to multiple databases
- WorkflowInvoker Enables the portal to invoke a business process

The custom controls are described in detail in the following sections.



JoinDB Custom Control

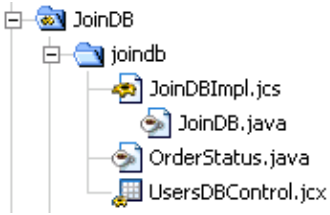
The JoinDB custom control coordinates database queries to multiple databases, including:

- Employee Information database used by the Employee Information component (described in Viewing Employee Information)
- Office Equipment Order Management database used by the Office Equipment Order Management component (described in Ordering Office Equipment)

The JoinDB custom control is employed by the CheckStatus action defined in the Order portlet page flow, as described in Order Portlet.

The JoinDB custom control and related files are stored within the JoinDB folder in the e2ePortal application, as shown in the following figure.

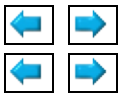
Files for the JoinDB Custom Control



The following table describes the contents of the JoinDB folder.

JoinDB Custom Control File Descriptions

File	Function
JoinDBImpl.jcs	Defines the implementation code for the custom control.
JoinDB.java	Defines the public interface for the custom control. This file is automatically generated by WebLogic Workshop.
OrderStatus.java	Defines the OrderStatus() method used to merge order tracking information (retrieved from the Office Equipment Order Management database) with employee information (retrieved from the Employee Information database). This method is used by the custom control.
UsersDBControl.jcx	Database control file that defines the database calls used to retrieve employee information. This control is defined in detail in Viewing Employee Information.



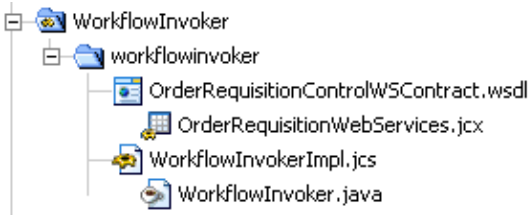
WorkflowInvoker Custom Control

The WorkflowInvoker custom control enables the portal to invoke a business process.

The WorkflowInvoker custom control is employed by the PlaceOrder action defined in the Order portlet page flow, as described in Order Portlet.

The WorkflowInvoker custom control and related files are stored in the WorkflowInvoker folder in the e2ePortal application, as shown in the following figure.

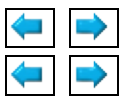
Files for the WorkflowInvoker Custom Control



The following table describes the contents of the WorkflowInvoker folder.

WorkflowInvoker Custom Control File Descriptions

File	Description
OrderRequisitionControlWSContract.wsdl	Web Service Definition Language (WSDL) file describing the methods and callbacks that the OrderRequisitionControlWS.jws web service implements, including method names, parameters, and return types. The OrderRequisitionControlWS.jws web service is defined in the e2eWorkflow application, as described in Reviewing the Office Equipment Order Management System.
OrderRequisitionWebServices.jcx	Process control that starts the Order Requisition business process through the OrderRequisitionControlWS.jws file. The OrderRequisitionControlWS.jws web service is defined in the e2eWorkflow application, as described in Reviewing the Office Equipment Order Management System.
WorkflowInvokerImpl.jcs	Defines the implementation code for the custom control. The invokeWorkflow() method calls the clientRequest() method defined by the OrderRequisitionControlWS.jws web service to invoke the Order Requisition workflow, defined in Order Requisition Business Process.
WorkflowInvoker.java	Defines the public interface for the custom control. This file is automatically generated by WebLogic Workshop.



Configuring Security in Web Applications

Secure access to the WebLogic Platform Tour is provided by WebLogic Server security realm user authentication.

To provide secure access to a Web application using user authentication, update the web.xml and weblogic.xml deployment descriptors to define the following parameters:

- User credentials
- Method of authentication
- Location of resources

Deployment descriptors are XML documents that describe the contents of an application directory or JAR file. The J2EE specification defines standard, portable deployment descriptors for J2EE components and applications, such as web.xml. BEA defines additional WebLogic−specific deployment descriptors, such as weblogic.xml, for deploying a component or application in a WebLogic Server environment.

In a WebLogic Server environment, the deployment descriptors are located in the WEB−INF directory of the Web application root directory, as required by the J2EE specification. The WEB−INF directory also contains two subdirectories for storing compiled Java classes and library JAR files. Note that the root directory of the Web application hierarchy defines the document root. All files under this root directory (except files in the WEB−INF directory) can be served to the client.

Security constraints, such as user authentication, are defined in the web.xml file.

For example, supposed you wanted to define security constraints for the e2ePortal application. Open the web.xml file for the e2ePortal application in WebLogic Workshop, as follows:

1. In the e2ePortal/e2ePortalProject folder, expand the **WEB−INF** directory in the Applications window.
2. Double−click the **web.xml** file, located in the WEB−INF directory.

   This file defines the security constraint information as follows:

```
<!−− Security −−>
<security−constraint>
<web−resource−collection>
<web−resource−name>login</web−resource−name>
<url−pattern>/Controller.jpf</url−pattern>
</web−resource−collection>
<auth−constraint>
<role−name>employee</role−name>
<role−name>manager</role−name>
</auth−constraint>
<user−data−constraint>
<transport−guarantee>NONE</transport−guarantee>
</user−data−constraint>
</security−constraint>
<security−constraint>
        <web−resource−collection>
                <web−resource−name>employee</web−resource−name>
                <url−pattern>/Employee.portal</url−pattern>
        </web−resource−collection>
        <auth−constraint>
                <role−name>employee</role−name>
        </auth−constraint>
        <user−data−constraint>
                <transport−guarantee>NONE</transport−guarantee>
        </user−data−constraint>
</security−constraint>
```

```
<security−constraint>
        <web−resource−collection>
                <web−resource−name>manager</web−resource−name>
                <url−pattern>/Manager.portal</url−pattern>
        </web−resource−collection>
        <auth−constraint>
                <role−name>manager</role−name>
        </auth−constraint>
        <user−data−constraint>
                <transport−guarantee>NONE</transport−guarantee>
        </user−data−constraint>
</security−constraint>
<login−config>
<auth−method>FORM</auth−method>
<form−login−config>
<form−login−page>/login/loginRedirect.jsp</form−login−page>
<form−error−page>/login/login_error.jsp</form−error−page>
</form−login−config>
</login−config>
<security−role>
<role−name>employee</role−name>
</security−role>
<security−role>
<role−name>manager</role−name>
</security−role>
<security−role>
<description>Administrator</description>
<role−name>Admin</role−name>
</security−role>
<security−role>
<description>all users</description>
<role−name>AnonymousRole</role−name>
</security−role>
```

The following table describes the XML elements in the previous code excerpt.

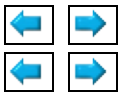
This element...	Performs this function...
<login−config>	Enables form−based authentication and specifies: • /login/loginRedirect.jsp as the JSP page containing the authentication form • /login/login_error.jsp as the JSP file page to return in the event of an error The login.jsp file that is initially displayed in the Log In portlet generates an input form for obtaining user login information. It defines the action as follows: `<% String formActionURL = response.encodeURL("j_security_check");%> <form method="post" action="<%formActionURL %>">` As a result, the login information is passed to j_security_check for authentication.
<security−constraint>	Specifies the following security details:

	• Web application component to which the security constraint is applied this is the first page that is called when you invoke the Web application in a browser. • User roles that have security access, such as employee and manager. • No transport guarantees are required for communications between client and server.
<security−role>	Specifies valid security roles. At deployment time, security role mappings are obtained from the weblogic.xml file. (This file is configured using the WebLogic Server Administration Console.)

To learn more about configuring security, see Securing Web Applications in *Programming WebLogic Security*, available on E−docs.

Before proceeding to the next step in the WebLogic Platform Tour:

1. Close all open files by choosing **File** > **Close Files**
2. To conserve screen real estate, temporarily minimize the WebLogic Workshop window.

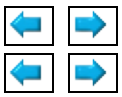


Viewing Employee Information

In this part of the Tour, you (new employee John Smith) view your employee information. Along the way you will learn how the application uses a built−in database control to simplify access to employee information from the Employee Information database.

Follow the steps described in Step Through the Tour and review the concepts described in this section to learn about:

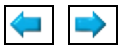
- Using a Built−In Control
- Reviewing the Employee Information System

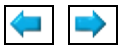


Step Through the Tour

Note: In Logging In to the Avitek Corporate Intranet, you logged in to the Avitek corporate intranet as John Smith, the new employee. Currently, you are viewing the Employee portal.

Your employee information is populated automatically when you log in. For example, you can view the employee profile for John Smith within the Employee portlet.





Using a Built−In Control

WebLogic Workshop provides a set of *built−in* controls that make it easy to access enterprise resources, such as Enterprise Java Beans (EJBs) and databases, from within your application. The control handles the work of connecting to the enterprise resource for you, so that you can focus on the business logic to make your application work.

Most of the built−in controls are customizable controls. That is, when you add a new one to a project, WebLogic Workshop generates a *Java control extension* (JCX) file that extends the control. In some cases, such as with the Database control or JMS control, you can customize the control by adding or editing methods defined in the JCX file. Others are customized for you, as with the EJB control, which is customized based on the EJB the control will be accessing.

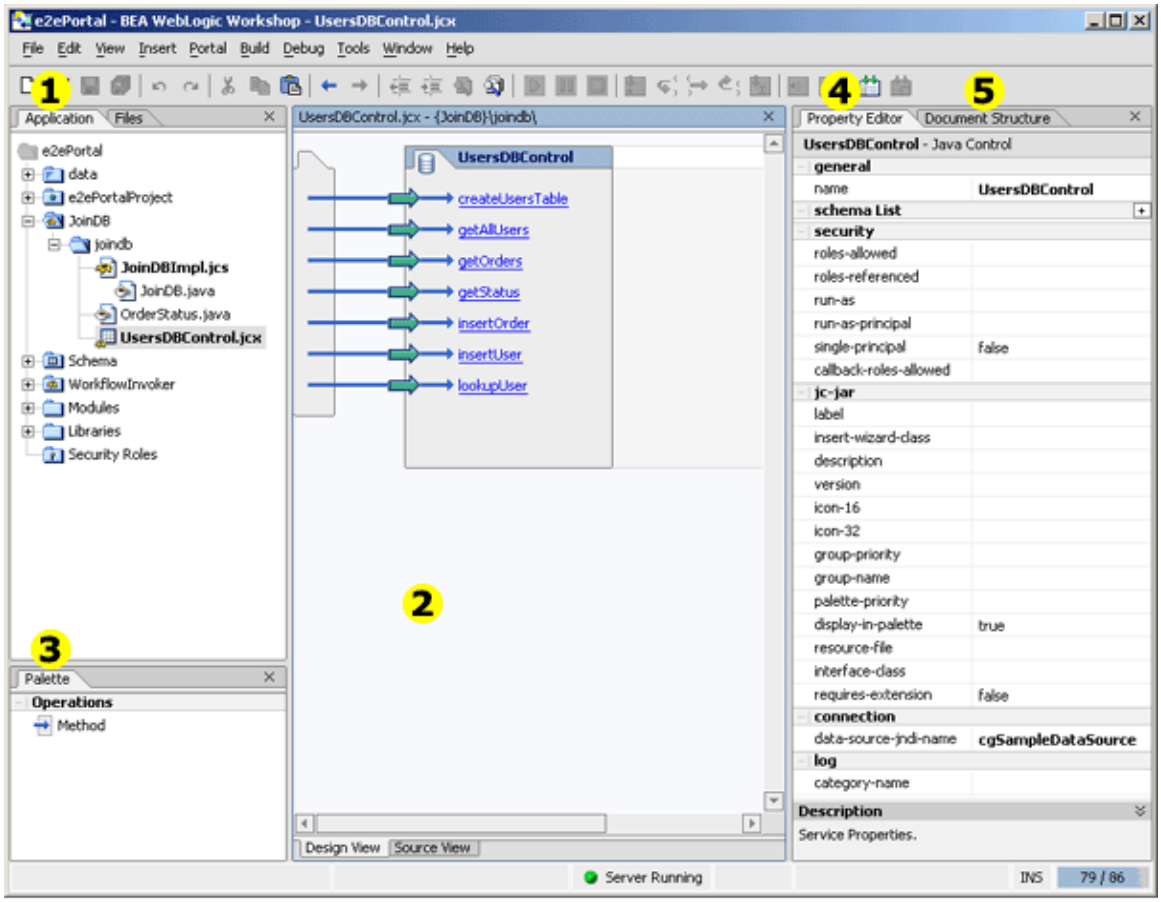
WebLogic Workshop makes it easy to use a built−in control: click Add from the Data Palette window when working in Design or Flow View, and then select the desired control from the drop−down list. You can create a control file:

- At the same time that you add a built−in control to your project. By default, WebLogic Workshop adds this control file with a .jcx extension to the same folder as the file that is currently open in Design View.
- Separately, and then point to it when adding the built−in control. For example, you can create a new database control file in the current folder by choosing Insert > Controls > Database. You are prompted to specify which built−in Java control you wish to extend and the parameters associated

with it.

The Control Editor is a graphical tool that enables you to design a control, such as the UsersDBControl database control shown in the following figure.

Control Editor



The following table describes the Control Editor tools shown in the previous figure.

Tools for Designing Controls

Callout #	Use this tool...	For this task...
1	Application Window	Create, view, and edit control files in your portal application projects. The names of control files end in .jcx.
2	Design and Source Views	

Design your control in this area.

3

Palette Window

Add methods, callbacks, variables, and so on, by dragging components from the Palette window and dropping them onto the Design View canvas. The options available in the Palette window depend on the type of control you are building.

4

Property Editor Window

Set properties for the currently selected control component. You can select a control component by clicking on it in the Design View canvas or by selecting its name in the Document Structure window (described below).

5

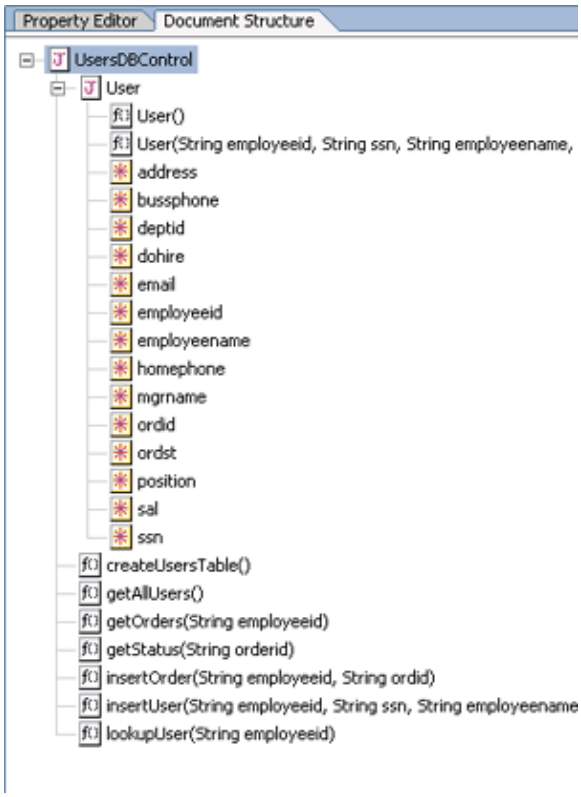
Document Structure Window

View a summary of the Java entities that comprise the control (see Figure: Document Structure Window for a Database Control). The summary includes the Java classes, methods and signatures, variables, and inner classes for the control.

You can select a Java entity in the Document Structure window by clicking on it. When you select a control method, it is selected in the Design View canvas and you can edit its properties in the Property Editor window. You can double−click on a Java entity to jump to its code location in Source View.

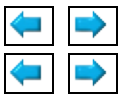
The following figure shows the Document Structure window for a database control.

Document Structure Window for a Database Control



To learn more about adding built−in controls:

- View a Demo, Building a Control, to learn how to build a database control.
- See Tutorial: Java Control in the *WebLogic Workshop Help*.
- See Using Built−In Java Controls in the *WebLogic Workshop Help*.



Reviewing the Employee Information System

When the Employee intranet portal is loaded, the UsersDBControl database control connects to the employee information database to retrieve the employee profile for the current user.

A *database control* is one example of a built−in Java control. This type of control makes it easy to access a relational database from your application. You simply issue SQL commands to the database and the database control performs the following tasks on your behalf:

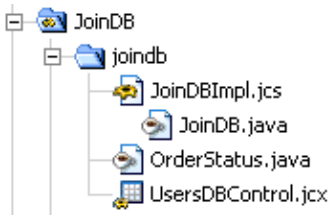
- Connects to the database you do not have to understand Java Database Connectivity (JDBC).
- Automatically performs the translation from database queries to Java objects simplifying access to the results.

When you add a new database control to your application, you specify the following:

1. Variable name for the control.
2. Whether you want to add a new database control file or reference an existing one.
3. Data source to which the control is bound (if you are creating a new control file in step 2).

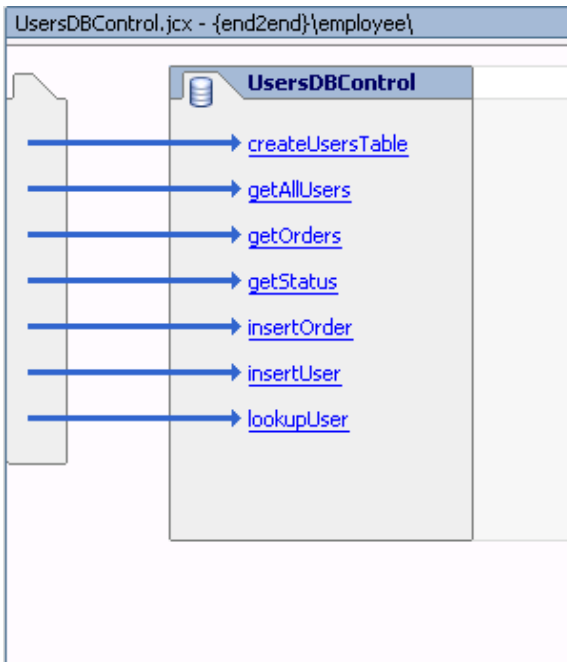
View the UsersDbControl database control in WebLogic Workshop by performing the following steps:

1. In the Application window, expand the e2ePortal/JoinDB/joindb folder, as shown:



2. Open the database control by double−clicking **UsersDBControl.jcx**.

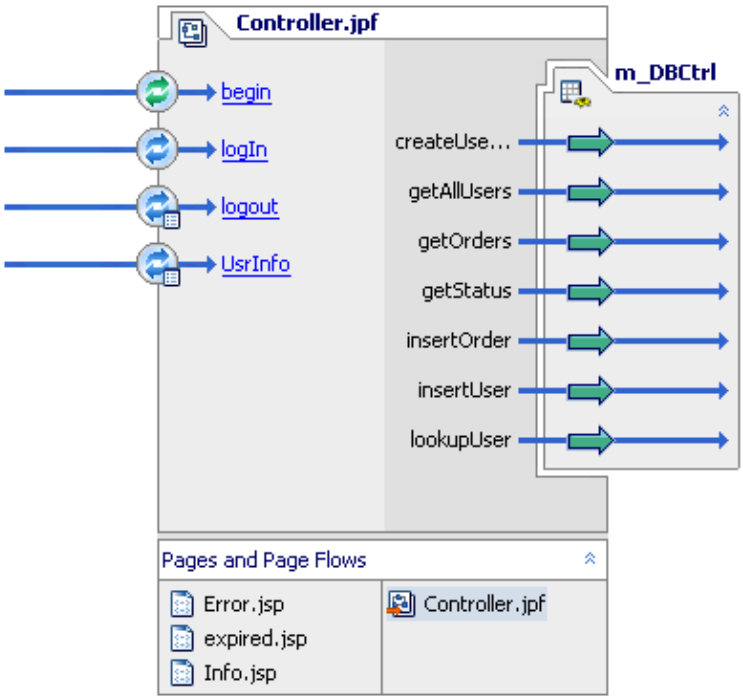
   The UsersDBControl database control and the methods associated with it are displayed, as follows:



   The database control provides a number of methods for accessing the employee information database.

3. Open the Employee page flow file by double−clicking **e2ePortalProject/employee/Controller.jpf**.
4. Select the **Action View** tab.

   An instance of the UsersDBControl database control, m_DBCtrl, and the methods associated with it are displayed on the right−hand side of the page flow.



5. Select the **Source View** tab.

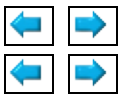
   The m_DBCtrl instance is used to look up the current user and populate the InfoForm form bean with the employee profile information.

```
HttpSession sess= getSession();
HttpServletRequest req=getRequest();
joindb.UsersDBControl.User user= null;
.
.
.
user = m_DBCtrl.lookupUser(empId);
.
.
.
if(user==null)
  return new Forward( "theFirstPage" );
else {
  sess.setAttribute("employeeid",user.employeeid);
  sess.setAttribute("employeename",user.employeename);
  sess.setAttribute("ssn",user.ssn);
  sess.setAttribute("departmentid",user.deptid);
  sess.setAttribute("mgrname",user.mgrname);
  sess.setAttribute("dateofhire",user.dohire);
  sess.setAttribute("salary",user.sal);
  sess.setAttribute("officeaddress",user.address);
  sess.setAttribute("homephone",user.homephone);
  sess.setAttribute("businessphone",user.bussphone);
  sess.setAttribute("position",user.position);
  sess.setAttribute("email",user.email);
}
return new Forward( "theFirstPage" );
}
.
```

```
.
.
protected Forward UsrInfo(InfoForm form)
{
        return new Forward("success");
}
```

Before proceeding to the next step in the WebLogic Platform Tour:

1. Close all open files by choosing **File** > **Close Files**
2. To conserve screen real estate, temporarily minimize the WebLogic Workshop window.



Ordering Office Equipment

In this part of the Tour, you play the roles of two Avitek employees in order to test−drive different functions of the company's Office Equipment Order Management system:

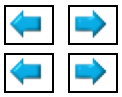
1. As the new employee, John Smith, you order a laptop for yourself, using Avitek's online Office Equipment Order Management system.
2. Then you assume the identity of John's manager, Rachel Burns, and approve John's laptop order.
3. Finally, as John Smith, you can check the status of the laptop order.

As shown by these steps, this section describes the Office Equipment Order Management process, demonstrating the integration of an application built using features of WebLogic Integration. Along the way you will learn how the business process management feature is used to integrate diverse applications and human participants, and to coordinate the exchange of information among application resources.

Note: Business process management is a feature of WebLogic Integration which is delivered as part of the WebLogic Workshop Platform Edition.

Follow the steps described in Step Through the Tour and review the concepts described in this section to learn about:

- Designing Business Processes Using WebLogic Integration
- Reviewing the Office Equipment Order Management System



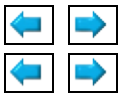
Step Through the Tour

When you log in to the Avitek corporate intranet using your employee ID, you can access the Office Equipment Order Management system and submit an order for office equipment from the Order portlet.

The Office Equipment Order Management system enables employees to order office equipment, and then monitors the progress of the order through each phase in the ordering process: order submittal, manager approval, inventory check, shipping, and billing.

Step through this part of the WebLogic Platform Tour to familiarize yourself with the order management process being demonstrated. Specific steps include:

- Step 1: Place a New Order
- Step 2: Approve an Order
- Step 3: Check the Status of an Order



Step 1: Place a New Order

As new employee John Smith, you order a new laptop.

Note: In Logging In to the Avitek Corporate Intranet, you logged in to the Avitek corporate intranet as John Smith, the new employee. Currently, you are viewing the Employee portal.

To submit your order, perform the following steps:

1. Click **Place a New Order** in the Order portlet.

   The Item ID form is displayed.

2. Enter **notebook_kit1** (the item ID for the desired laptop) in the Item ID field to request a new laptop.

   Note: You can also enter desktop_kit1 as a valid option.

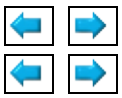
3. Click **Place a New Order** to place the order.

   The Order form is submitted and a notification is sent to Rachel Burns, John Smith's manager. The Order portlet is updated to indicate that the order was successfully placed.

4. View the status of the order by clicking **Check Order Status**.

   The Order portlet is updated with the current status of the order.

5. Click **Log Out** in the Log Out portlet to log out of the corporate intranet.
6. Return to the Avitek Log In window by clicking **Log In Again** in the Log Out portlet.



Step 2: Approve an Order

As John's manager, Rachel Burns, you approve John's order:

1. Log in to the Avitek corporate intranet as Rachel Burns, using the following user name and password:

   User Name: **rachel**
   Password: **emanager**

   The manager's view of the Avitek intranet portal is displayed giving you access to the Worklist user interface. The Worklist user interface displays all pending order for Rachel's direct employees.

   Note: You can also view a list of direct reports in the Direct Reports portlet.

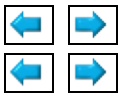
2. Select the pending equipment approval for John Smith's laptop.
3. Select **Respond to the Task** to approve the order for the new laptop.

   The Task Response text box is displayed.

4. Type **Approved** in the text box.
5. Click **OK**.

   Notice that the order has been removed from the list of pending tasks.

6. Click **Log Out** in the Log Out portlet to log out of the corporate intranet.
7. Return to the Avitek Log In screen by clicking **Log In Again** in the Log Out portlet.



Step 3: Check the Status of an Order

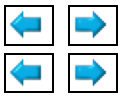
You can check the status of an order to track its progress, as it passes to the Check Inventory, Shipping, and Billing business processes.

1. Log in to the Avitek corporate intranet as John Smith, using the following user name and password:

   User Name: **john**
   Password: **employee**

2. In the Order portlet, click **Check Order Status**.

   The Order portlet is updated with the current status of all pending orders.



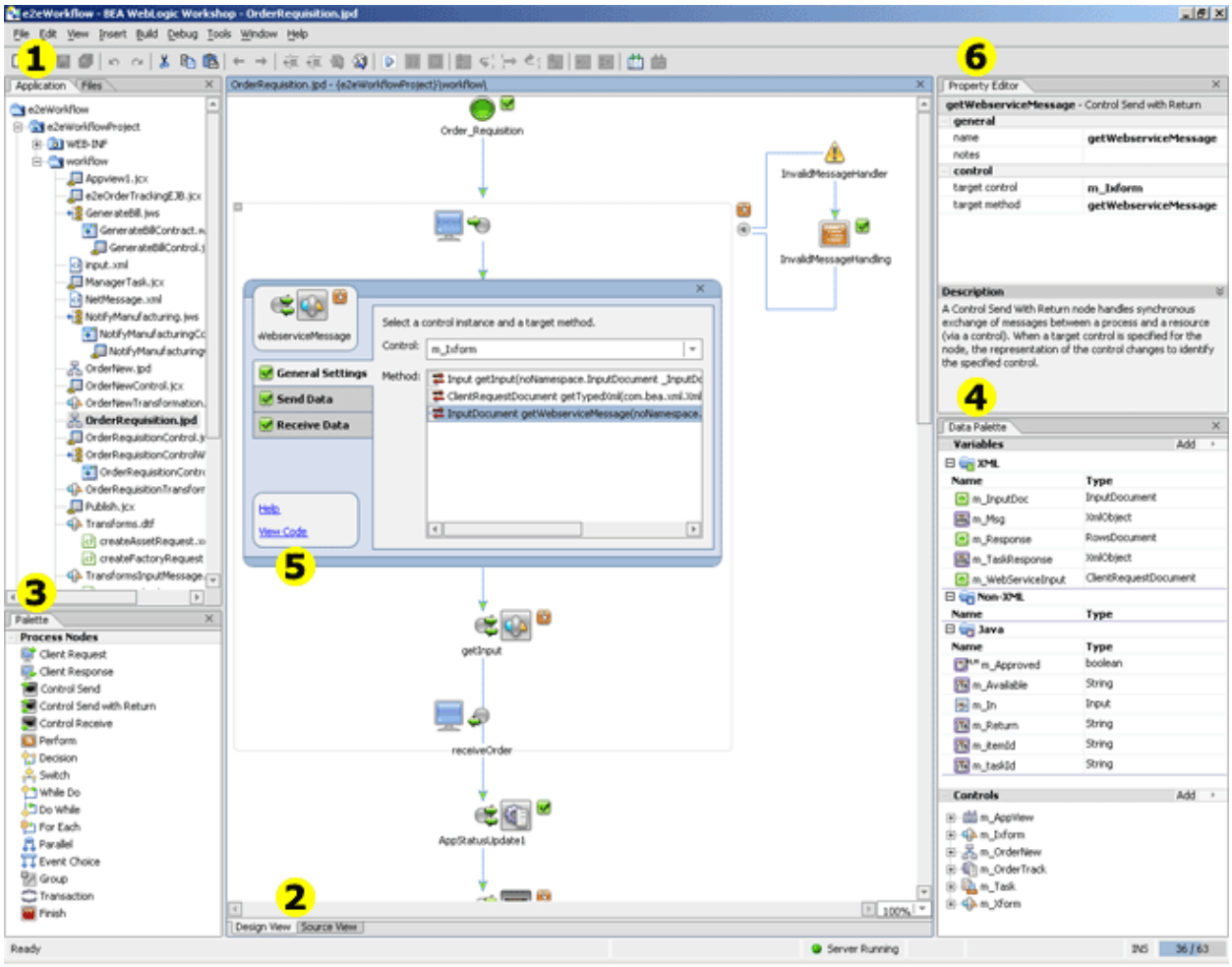
Designing Business Processes Using WebLogic Integration

WebLogic Integration simplifies the development, management, and delivery of business processes. A business process orchestrates the execution of business logic and the exchange of business documents among applications, users, enterprise networks, and trading partners in a loosely coupled fashion.

WebLogic Workshop makes it easy to create a business process: click the target folder and choose File > New > Process File. The Process Designer opens and displays a new process file that contains a Start and Finish node.

The Process Designer is a graphical tool that enables you to design business processes and establish interactions with clients and resources, such as databases, JMS queues, file systems, and so on.

Process Designer



The following table summarizes the business process design tools shown in the previous figure.

Tools for Designing Business Processes

Callout #	Use this tool...	For this task...
1	Application Window	Create, view, and edit business process files in your application project. This window also provides shortcuts for performing specific operations on application files. For example, you can generate a Task control by right−clicking the business process (.jpd) file and selecting Generate Task Control File from the drop−down menu.
2	Design and Source Views	Design your business process in this area.

Switch between the Design and Source Views, as required. Changes made in one view are automatically reflected in the other.

3

Palette Window

Graph a set of nodes that depict your business process by dragging components from the Palette window and dropping them onto the Design View canvas. Node types include:

- Client Request Invoke a business process from a client so you can perform one or more operations using the methods exposed by the business process.
- Client Response Send a message back to the calling client.
- Control Send Interact asynchronously with an enterprise resource via a control.
- Control Send with Return Interact synchronously with an enterprise resource via a control.
- Control Receive Create a handler for a callback from a control.
- Perform Execute your custom Java code.
- Decision Select a path of execution based on the evaluation of one or more conditions.
- Switch Select a path of execution based on the evaluation of an expression specified on a condition node.
- While Do Evaluate a condition before the activities in a loop are performed.
- Do While Perform the activities in a loop before evaluating a condition.
- For Each Execute one or more activities that you specify in one or more nodes in the loop.
- Parallel Execute multiple activities simultaneously.
- Event Choice Wait to receive multiple events before proceeding.
- Finish End execution.

4

Data Palette Window

Define instances of variables and Java controls to be used in the business process.

5

Node Builder

Design the functionality of the business process node. To invoke the node builder, double−click the node.

A node builder is a task−driven interface that enables you to define the logic for the nodes in the business process. Examples of this logic includes:

- Methods that are invoked by clients to start your business process or by which the business process responds to clients.
- Controls that your business process node uses to interact with resources and the data exchanged in the interaction.
- Data transformations to map heterogeneous data as it is exchanged between your business process and resources.

6

Property Editor Window

Set properties for the nodes in your business process.

The WebLogic Integration Administration Console, shown in the following figure, allows administrators to manage and monitor the resources required for your WebLogic Integration applications, such as business processes.

WebLogic Integration Administration Console

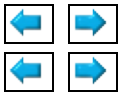
The following lists the specific resources that can be managed by the WebLogic Integration Administration Consoles:

- Process Instance Monitoring Monitor instances of business processes.
- Process Configuration Configure business processes.
- Message Broker Monitor Message Broker message−based communication.

- Event Generators Create, view, and edit event generators.
- Worklist Administration Administer and monitor worklist task instances.
- Application Integration Monitor enterprise adapters.
- Trading Partner Management Configure and manage trading partners.
- System Configuration Configure system and security information.
- User Management Configure users and roles that access integration system resources.
- Business Calendar Configuration Create and edit the business calendars that determine user availability or timing of system events.

To learn more about designing and managing business processes in WebLogic Workshop:

- View a demo, Designing a Business Process, that shows how to design a business process using the business process design tools.
- See Guide to Building Business Processes in the *WebLogic Workshop Help*.
- See Process Configuration, in *Managing WebLogic Integration Solutions*, available on E−docs.



Reviewing the Office Equipment Order Management System

The e2eWorkflow application defines the Office Equipment Order Management system, a WebLogic Integration business process management application that directs the lifecycle of an order for office equipment.

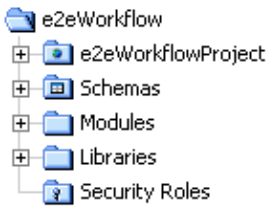
To view the WebLogic Integration business process management application in WebLogic Workshop:

1. Open the e2eWorkflow application, as follows:
    a. Choose **File** > **Open** > **Application**.
    b. Navigate to the samples\platform\end2end\e2eWorkflow folder of the WebLogic Platform installation.
    c. Select **e2eWorkflow.work**.

    The names of application files end in .work.

    d. Click **Open** to open the application.
2. If the application files are not currently displayed in WebLogic Workshop, choose **View** > **Application**.

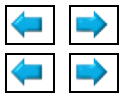
The e2eWorkflow application contents are displayed, as follows.



The following table describes each of the e2eWorkflow application folders.

e2eWorkflow Application Folders

Folder	Contents
e2eWorkflowProject	Directories and files for WebLogic Integration business process management application, including: • WEB−INF Standard Web application folder containing deployment descriptors and validation files. • workflow Folder containing WebLogic Integration business process management application files. The contents of this folder are described in detail in the sections that follow. Note that a sample test file, input.xml, is provided that enables you to invoke the Order Management business process outside of the portal, directly from a web service. Use of this file is not demonstrated in the WebLogic Platform Tour. If you wish to call this file to invoke the business process from a web service, you must edit the file to specify a unique order ID and valid item ID for the <orderId> and <itemId> fields, repsectively.
Schemas	XML and WebLogic Integration Application Integration (WLAI) schema definitions, and application view information used by the Application View control.
Modules	Application View and Order Tracking EJB JAR files. This folder is standard in all application projects. You can modify the contents based on your application requirements.
Libraries	Run−time related library files. This folder is standard in all application projects. You can modify the default contents based on your application requirements.
Security Roles	Security role information. This folder is standard in all application projects. You can modify the default contents based on your application requirements. For the e2eWorkflow application, this folder is empty.



Overview of the Office Equipment Order Management Business Processes

The e2eWorkflow application business process consists of several nested business processes, as defined in the following table.

Business Processes for Office Equipment Order Management System

This business process...	Defines how...

Order Requisition

The entire ordering process works, specifically: how an order for office equipment is submitted and approved by a manager; how the inventory is checked; and how the factory is notified if the ordered item is out of stock.

Order New

A new item is ordered from the factory.

Billing

The seller bills the buyer for payment.

Shipping

The ordered item is scheduled for delivery.

The business processes interact with required resources using built−in Java controls. The following table defines the types of controls used by the Office Equipment Order Management business processes, and the Java control extension (JCX) files defined for the sample application.

Note: As noted previously, for some built−in controls, when you add the control to your application you must create or reference a JCX file in your application project to define the functionality of the control. To learn more about creating a JCX file for a built−in control, see Using a Built−In Control.

Built−In Controls Used in the Office Equipment Order Management Business Processes

This Control...	**Simplifies...**	**For example...**
Application View	Access to an enterprise application using an Application View. An Application View must be created using the Application View Console before it can be referenced using an Application View control.	The procedure for building an Application View is outside the scope of the WebLogic Platform Tour. To learn more, see Overview: Application Integration in the *WebLogic Workshop Help*. Appview1.jcx is an application view control that communicates with the company and factory inventory databases through an the Application Integration (AI) Application View called e2eWorkflow_AppView1_ApplicationView−ejb.jar. This Application View is imported into the Modules project folder. Use of this control is illustrated in the Order Requisition and Order New business processes.
EJB	Access to an EJB deployed from your application.	

The procedure for building an EJB is outside the scope of the WebLogic Platform Tour. To learn more, see Developing Enterprise Java Beans in the *WebLogic Workshop Help*.

e2eOrderTrackingEJB.jcx is an EJB control that tracks the status of the order throughout the process by interfacing with the Order Tracking EJB called E2EOrderTrackingEJB.jar. This EJB is imported into the Modules project folder.

Use of this control is illustrated in all business processes.

Message Broker

Use of publish−and−subscribe functionality that is available with Message Broker, a message−based communication protocol that includes a powerful message filtering capability.

Publish.jcx is a Message Broker control that communicates with the Billing and Shipping business processes.

Use of this control is illustrated in the Order New business process.

Process

Access to a business process in your project.

OrderRequisitionControl.jcx is a process control that starts the Order Requisition business process through the OrderRequisitionControlWS.jws file.

Use of this control is illustrated in the Order Requisition business process.

OrderNewControl.jcx is a process control that starts the Order New business process.

Use of this control is illustrated in the Order New business process.

Task

Access to a Worklist user interface, allowing your business process to manage work items and handle callbacks via the WebLogic Integration Worklist user interface.

ManagerTask.jcx is a task control that sends an asynchronous message to Rachel Burns, John Smith's manager, for approval of an office equipment order.

Use of this control is illustrated in the Order Requisition business process.

Transformation

Mapping and conversion of data from one format to another. In this way, transformation logic is encapsulated within the transformation control and modifications to that logic do not impact the process definition.

Transforms.dtf is a transformation control that enables communication with the company and factory inventory databases when the inventory is checked.

Use of this control is illustrated in the Order Requisition business process.

TransformsInputMessage.dtf is a transformation control that enables communication between business processes for an incoming order.

Use of this control is illustrated in all business processes.

TransformsNetworkMessage.dtf is a transformation control that enables communication between the Message Broker and the Billing and Shipping business processes. The Message Broker uses an intermediary format to deliver a loosely−coupled system.

Use of this control is illustrated in the Order New, Billing, and Shipping business processes.

Web Service

Access to any web service that publishes a WSDL (Web Service Definition Language) file. A WSDL file describes the methods and callbacks that a web service implements, including method names, parameters, and return types.

The procedure for building a web service is outside the scope of the WebLogic Platform Tour. For more information, see Building Web Services in the *WebLogic Workshop Help*.

NotifyManufacturingControl.jcx is a web service control that communicates orders to the Manufacturing department through the NotifyManufacturing.jws web service.

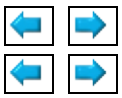
Use of this control is illustrated in the Order New business process.

GenerateBillControl.jcx is a web service control that generates a bill for the ordered item through the GenerateBill.jws web service.

Use of this control is illustrated in the Billing business process.

The following sections step through each business process in detail by following the path of the nodes:

- Order Requisition Business Process
- Order New Business Process
- Billing Business Process
- Shipping Business Process



Order Requisition Business Process

The Order Requisition business process defines the overall sequence of events that occur when someone at Avitek orders office equipment. The business process is started when an employee submits an order for office equipment. It proceeds as follows:

1. Employee submits the order for manager approval.

   If the order is approved by the manager, the business process proceeds to the next step. If the order is rejected by the manager, the business process ends the ordering process.

2. The process checks the current inventory for the ordered item.

   If the ordered item is available in the current inventory, the business process ends the ordering process. If the ordered item is not available in the current inventory, the business process invokes the New Order business process.

To view the Order Requisition business process in WebLogic Workshop, double−click the **workflow/OrderRequisition.jpd** file in the Application window.

As you step through each node description, perform the following steps:

1. Double−click the node in the business process to invoke the node builder and display details about the node.
2. Select **View Code** in the node builder (or by right−clicking the node), when applicable, to display the related code in the Source View window.

The following table describes each node in the Order Requisition business process.

Stepping Through the Order Requisition Business Process

Business Process Node	Description
 receiveOrder	Defines the event that starts the business process: an employee submits an order for office equipment. When you specify that the business process is Invoked via a Client Request with Return option, your Start node is displayed in expanded format. You can collapse the node view by clicking on the minus sign in the upper left−hand corner. To receive the order, the Client Request node calls the clientRequest() method that is exposed by the OrderRequisitionControlWS.jws web service. The web service uses the OrderRequisitionControl process control to invoke the business process. The order is passed as an XMLObject argument and the data is transformed to a ClientRequestDocument type using the data mapping tool. The Client Response node calls the clientReturn() method to return the results to the calling process. **Note:** For the WebLogic Platform Tour, the OrderRequisitionControlWS.jws web service was generated separately from the business process in order to decouple to business process and portal client applications. In addition, WebLogic Workshop enables you to generate a web service from a business process: right−click the business process and choose Generate WSDL File. Then, right−click the WSDL file and choose Generate Web Service.
 getWebserviceMessage	

Transforms the message to the appropriate format.

The Control Send with Return node uses m_Ixform, an instance of the TransformsInputMessage.dtf data transformation control, to call the getWebserviceMessage() method to transform the message received from the ClientRequestDocument type to the InputDocument type.

Transforms the message to the appropriate format.

The Control Send with Return node uses m_Ixform, an instance of the TransformsInputMessage.dtf data transformation control, to call the getInput() method to transform the message received from the InputDocument type to the Input type.

Defines the method used to return the results to the calling process.

The Client Response node calls the clientReturn() method to return the results to the calling process as a String argument.

Defines the exception handler for the Client Request node.

The Perform node calls the invalidMessageHandler() method to catch and handle an exception using your custom business logic.

It is recommended that you add proper exception handlers, as required by your application. In particular, they should be added to the Start nodes in your business processes.

Updates the status of the order.

The Control Send with Return node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the create() method to create a new order tracking ID and update its status to indicate that the order has been received.

Sends an asynchronous message to Rachel Burns, requesting approval of the order for a new laptop.

The Control Send node uses m_Task, an instance of the ManagerTask.jcx task control, to call the createTaskByName() method to create the new task.

Sends an asynchronous message to Rachel Burns, requesting approval of the order for a new laptop.

The Control Send node uses m_Task, an instance of the ManagerTask.jcx task control, to call the assignTaskToUser() method to assign the task to Rachel Burns.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the task has been assigned to the manager.

Receives the approval response from Rachel Burns.

The Control Receive node uses m_Task, an instance of the ManagerTask.jcx task control, to call the onTaskCompleted() method to return the response from Rachel Burns.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the task has been approved by the manager.

Branches the business process flow, based on whether or not the order is approved by the manager. If the order is approved, then the business process proceeds to the Check Inventory node. If the order is rejected,

then the order process ends.

The Decision node calls the isApproved() Java method to branch the business process flow accordingly.

Checks the company inventory for the ordered item.

The Control Send node uses m_AppView, an instance of the Appview1.jcx Application View control, to call the CheckInventoryAsset() method to interface with the Application View to check the company inventory.

The node also uses m_Xform, an instance of the Transforms.dtf data transformation control, to call the following methods:

- createAssetRequest() to transform the request received into the appropriate format.
- isAvailable() to determine if the item is available in the company inventory.

Branches the business process flow, based on whether or not the ordered item is available in the company inventory. If the item is available, then the business process proceeds to the UpdateAppStatusAvailability node. If the item is not available, then the business process proceeds to the Not Available node.

The Decision node calls the isAvailable() Java method to branch the business process flow accordingly.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order is available in the local inventory.

Branches the business process flow, based on whether the reason the ordered item is unavailable is lack of inventory or another condition. If the item is unavailable due to lack of inventory, then the business process proceeds to the UpdateAppStatusNonAvailability node. If the item is unavailable due to another reason, then the order process ends.

The Decision node calls the isNotAvailable() method to branch the business process flow accordingly.

Updates the status of the order.

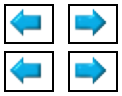
The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order is not available in the local inventory.

Starts the New Order business process to request the item that is out of stock.

The Control Send node uses m_OrderNew, an instance of the OrderNewItemControl.jcx process control, to call the receiveNewOrderRequest() method to start the New Order business process.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the Order New business process has been invoked.



Order New Business Process

The Order New business process defines the process for requesting a new item from the factory inventory, if the ordered item is not available in the company inventory.

The business process is started by the Order Requisition business process if the item is not available in the company inventory. The business process checks the factory inventory for the item. If the item is available in the factory inventory, the business process ends the ordering process. If the item is not available in the factory inventory, the business process notifies the Manufacturing department to produce a new item and initiates the Billing and Shipping business processes.

To view the Order New business process in WebLogic Workshop, double−click the **workflow/OrderNew.jpd** file in the Application window.

As you step through each node description:

1. Double−click the node in the business process to invoke the node builder and display details about the node.
2. Select **View Code** in the node builder (or by right−clicking the node), when applicable, to display the related code in the Source View window.

The following table describes each node in the Order New business process.

Stepping Through the Order New Business Process

Business Process Node	Description
 OrderNewReceived	Defines the asynchronous event that starts the business process: a call from the Order Requisition business process indicating that the ordered item is out of stock. The Client Request node receives the order as an InputDocument argument.
 getInput	Transforms the message to the appropriate format. The Control Send with Return node uses m_Ixform, an instance of the TransformsInputMessage.dtf data transformation control, to call the getInput() method to transform the message received from the InputDocument type to the Input type.
 CheckFactoryInventory	Checks the factory inventory for the ordered item. The Control Send node uses m_AppView, an instance of the Appview1.jcx Application View control, to call the CheckInventory() method to interface with the Application View to check the factory inventory. The node also uses m_xform, an instance of the Transforms.dtf data transformation control, to call the following methods: • createFactoryRequest() to transform the request received into the appropriate format. • isFactoryAvailable() to determine if the item is available in the factory inventory.
 findByPrimaryKey	Retrieves the order tracking ID for the current order. The Control Send with Return node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the findByPrimaryKey() method to locate the order tracking ID for the current order.

Branches the business process flow, based on whether or not the ordered item is available in the factory inventory. If the item is available, the business process proceeds to the UpdateAppStatusAvailability node;. If the item is not available, then the business process proceeds to the Not Available node.

The Decision node calls the isAvailable() Java method to branch the business process flow accordingly.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order is available in the local inventory.

Branches the business process flow, based on whether the reason the ordered item is unavailable is lack of inventory or another condition. If the item is unavailable due to lack of inventory, then the business process proceeds to the UpdateAppStatusNonAvailability node. If the item is unavailable due to another reason, the order process ends.

The Decision node calls the isNotAvailable() method to branch the business process flow accordingly.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order is not available in the local inventory.

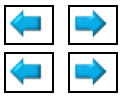
Notifies the Manufacturing department that a new item needs to be produced.

The Control Send node uses notifyManufacturingControl, an instance of the NotifyManufacturingControl.jcx web service control, to call the notifyMethod() method to notify the Manufacturing department.

Requests the Billing and Shipping departments to generate a bill for the new item (just produced by the Manufacturing department) and to ship it to the recipient.

The Control Send node uses mPublish, an instance of the Publish.jcx Message Broker control, to call the notifyMethod() method to send requests to the Billing and Shipping departments.

Before sending the requests, the node uses m_NetTransform, an instance of the TransformsNetworkMessage.dtf data transformation control, to call the transformToBasicTypedNet() method to transform the requests to an intermediary format, NetMessageDocument, in preparation for delivery to the Message Broker.



Billing Business Process

The Billing business process defines the sequence of tasks required to bill for the ordered item.

The business process is started by the Order New business process if the item needs to be produced by the Manufacturing department. The business process checks the origin of the order. If the order was generated inside the company, the business process charges the appropriate department. If the order was generated outside of the company, the business process generates a bill and sends it to the recipient.

To view the business process in WebLogic Workshop, double−click the **workflow/WorkflowBilling.jpd** file in the Application window.

As you step through each node description:

1. Double−click the node in the business process to invoke the node builder and display details about the node.
2. Select **View Code** in the node builder (or by right−clicking the node), when applicable, to display the related code in the Source View window.

The following table describes each node in the Billing business process.

Stepping Through the Billing Business Process

Business Process Node	Description
	Defines the asynchronous event that starts the business process: a billing−related message is received. The Client Request node uses mPublish, an instance of the Publish.jcx Message Broker control, to call the subscription() method to subscribe to the Message Broker, specifying end2end/billingshipping as the Channel

name. The node uses a filter, comparing the destination1 field in the NetMessageDocument argument received to the filter value, billing.

Transforms the message from the Message Broker intermediary format, NetMessageDocument.

The Control Send with Return node uses m_netTransforms, an instance of the TransformsNetworkMessage.dtf data transformation control, to call the transformFromBasicNet() method to transform the message received from the NetMessageDocument type, used by the Message Broker, to the InputDocument type.

Transforms the message to the appropriate format.

The Control Send with Return node uses m_transformsInput, an instance of the TransformsInputMessage.dtf data transformation control, to call the getInput() method to transform the message received from the InputDocument type to the Input type.

Retrieves the order tracking ID for the current order.

The Control Send with Return node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the findByPrimaryKey() method to locate the order tracking ID for the current order.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order has been received by the Billing department.

Branches the business process flow, based on whether or not the order was generated inside the company. If the order is internal, the business process proceeds to the Charge Dept node. If the order is external to the company, then the business process proceeds to the Not Intra Company node.

The Decision node calls the isIntra() method to branch the business process flow accordingly.

Charges the appropriate department to process an order that is internal to the company.

The Perform node calls the chargeDept() Java method to charge the appropriate department.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the department has been charged.

Branches the business process flow, based on whether or not the order was generated external to the company. If the order is external, the business process proceeds to the GenerateBill node. If some other condition is raised, the order process ends.

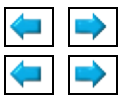
The Decision node calls the isNotIntra() method to branch the business process flow accordingly.

Generates a bill for the order.

The Control Send node uses billCtrl, an instance of the GenerateBillControl.jcx web service control, to call the generateBill() method to generate a bill for the order.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the bill has been generated.



Shipping Business Process

The Shipping business process defines the sequence of actions required to deliver an ordered item.

The business process is started by the Order New business process if the ordered item needs to be produced by the Manufacturing department. The business process schedules the order for delivery and generates a shipping notification.

To view the business process in WebLogic Workshop, double−click the **workflow/WorkflowShipping.jpd** file in the Application window.

As you step through each node description:

1. Double−click the node in the business process to invoke the node builder and display details about the node.
2. Select **View Code** in the node builder (or by right−clicking on the node), when applicable, to display the related code in the Source View window.

The following table describes each node in the Shipping business process.

Stepping Through the Shipping Business Process

Business Process Node	Description
	Defines the asynchronous event that starts the business process: a shipping−related message is received. The Client Request node uses mPublish, an instance of the Publish.jcx Message Broker control, to call the subscription() method to subscribe to the Message Broker, specifying end2end/billingshipping as the Channel name. The node uses a filter, comparing the destination2 field in the NetMessageDocument argument received to the filter value, shipping.
	Transforms the message from the Message Broker intermediary format, NetMessageDocument. The Control Send with Return node uses m_netTransforms, an instance of the TransformsNetworkMessage.dtf data transformation control, to call the transformFromBasicNet() method to transform the message received from the NetMessageDocument type, used by the Message Broker, to the InputDocument type.

Transforms the message to the appropriate format.

The Control Send with Return node uses m_transformsInput, an instance of the TransformsInputMessage.dtf data transformation control, to call the getInput() method to transform the message received from the InputDocument type to the Input type.

Retrieves the order tracking ID for the current order.

The Control Send with Return node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the findByPrimaryKey() method to locate the order tracking ID for the current order.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order has been received by the Shipping department.

Schedules the delivery of the ordered item.

The Perform node calls the schedule() Java method to schedule the delivery.

Updates the status of the order.

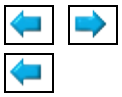
The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that the order has been scheduled for shipping.

Generates a shipping notice for the ordered item.

The Perform node calls the shippingNotification() Java method to generate the shipping notice.

Updates the status of the order.

The Control Send node uses m_OrderTrack, an instance of the e2eOrderTrackingEJB.jcx EJB control, to call the setDescRemote() method to indicate that a shipping report has been generated.



Learning More About WebLogic Platform

Congratulations on completing the WebLogic Platform Tour!

To learn more about using features provided by WebLogic Platform, refer to the resources listed in the following table.

Note: You can access the *WebLogic Workshop Help*, as follows:

- In WebLogic Workshop, choose Help > Help Topics.
- Navigate to e−docs at: http://e−docs.bea.com/workshop/docs81/index.html

Resources for More Information

To Learn More About Using...	See...
WebLogic Platform	WebLogic Platform documentation at http://e−docs.bea.com/platform/docs81/index.html
WebLogic Workshop to develop web services, business controls, and page groups, including tutorials and samples	◊ WebLogic Workshop Help ◊ Tutorials in the *WebLogic Workshop Help* ◊ Samples in the *WebLogic Workshop Help*
WebLogic Portal to develop and manage portals and portal resources, including tutorials and samples	◊ Building Portal Applications in the *WebLogic Workshop Help* ◊ Portal Tutorials in the *WebLogic Workshop Help* ◊ Portal Samples in the *WebLogic Workshop Help* ◊ WebLogic Administration Portal Help available on E−docs
WebLogic Integration to develop business processes for business process management, including tutorials and samples	◊ Building Integration Applications in the *WebLogic Workshop Help* ◊ Managing WebLogic Integration Solutions available on E−docs
WebLogic Server to develop J2EE applications, including tutorials and samples	WebLogic Server documentation at http://e−docs.bea.com/wls/docs81/index.html
WebLogic JRockit SDK to develop and run applications using the Java programming language	WebLogic JRockit documentation at http://e−docs.bea.com/wljrockit/docs81/index.html