

IDE User Guide

BEA Workshop for WebLogic Platform (Workshop for WebLogic) is a full-featured IDE for enterprise application development (SOA, J2EE, web applications). Workshop for WebLogic is based on the Eclipse framework.

Current Release Information:

- [What's New in 9.2](#)
- [Upgrading from 8.1](#)

Useful Links:

- [Tutorials](#)
- [Tips and Tricks](#)

Other Resources:

- [Online Docs](#)
- [Dev2Dev](#)
- [Discussion Forums](#)
- [Development Blogs](#)

Topics Included in This Section

Tutorial: Getting Started with BEA Workshop for WebLogic Platform

Introductory tutorial that takes you through the basics of the IDE: navigation, features, help, as well as the process of creating and testing a "Hello, world!" application.

Applications and Projects

Description of how to assemble enterprise applications from projects/modules.

Managing Project Dependencies

Description of how to set up dependencies between EAR projects and their web and EJB/utility projects.

Understanding the Build Process

Discussion of how to build your files before deployment.

Before you Begin: Setting Up the Server

Description of how to set up a server for development and testing.

Managing Servers

Description of how to manage development/testing servers.

Deploying, Running, and Debugging Applications

Explanation of how to deploy, run and debug enterprise applications.

Setting up Logging

Discussion of WebLogic Server message logging.

Working with Source Control

Description of how to check workspaces and projects into source control.

Exporting Archives

Description of how to export projects as EARs, WARs, or JARs.

Creating Custom Ant Build Files for Applications

Description of how to create Ant build files for your projects.

Setting Up Logging

Description of how to set up logging filters.

Tips and Tricks

Discussion of shortcuts and techniques to improve your productivity when using Workshop for WebLogic.

Troubleshooting

Techniques for resolving errors.

General IDE Dialogs

Describes project-related dialogs.

Tutorial: Getting Started with BEA Workshop for WebLogic Platform

BEA Workshop for WebLogic Platform (Workshop for WebLogic) is a set of plug-ins to the Eclipse IDE platform that allows you to quickly and easily create enterprise applications (SOA, J2EE) for deployment on BEA WebLogic Server.

Note: This tutorial requests that you create a new workspace; if you already have a workspace open, switching workspaces will restart the IDE. Before beginning, you might want to launch help in standalone mode to avoid an interruption the restart could cause, then locate this topic in the new browser. See [Using Help in a Standalone Mode](#) for more information.

This tutorial provides an introduction to using Workshop for WebLogic including:

- Understanding the basics, including the screen layout, editors, views, tools, menus and indicators
- Setting up the framework of projects, folders and files to contain a J2EE application
- Creating a simple "Hello, world!" application
- Starting a WebLogic Server
- Testing your application by running it on the WebLogic Server

If you have used Eclipse with other plug-ins or with the Java Development Tools (JDT) plug-in, you will find this material very familiar.

Background Information

This tutorial assumes a good working knowledge of Java and object-oriented programming as well as a basic understanding of web applications.

The Eclipse IDE Environment

Workshop for WebLogic is built on the Eclipse IDE framework. Eclipse is an open source initiative that is widely supported in industry. As a result, many of the standard features of Workshop for WebLogic are described in the Eclipse documentation, available at <http://eclipse.org>. Familiarity with Eclipse is not required for this tutorial, but basic Eclipse knowledge is helpful.

Programming Tools and Frameworks

Workshop for WebLogic makes extensive use of Java 5 annotations. You can learn more about annotations at <http://java.sun.com>.

This tutorial does not assume expertise in J2EE. However if you are unfamiliar with J2EE, more information is available at <http://java.sun.com>.

Workshop for WebLogic integrates the Beehive open source framework, a project of the Apache Software Foundation (<http://beehive.apache.org>). Beehive provides tools for creating *page flows* (JSP files linked to a Java controller for maintaining user state information) and *controls* (an object model for standardized simplified access to resources and encapsulated business logic). A local copy of the [Apache Beehive documentation](#) is included with your Workshop for WebLogic installation.

Tutorial Overview

To demonstrate the features of Workshop for WebLogic, this tutorial will walk you through the process of creating and running a simple "Hello, world!" application. At each point, the tutorial will discuss the features of the Workshop for WebLogic interface and show you how to access other resources that will help you learn more about Workshop for WebLogic. This tutorial is geared to all levels of users, so users with previous experience of Eclipse may want to skim this material and then proceed directly to the more specific tutorials, listed in [Related Topics](#) below.

Steps in this Tutorial

[Exploring the Features of Workshop for WebLogic](#)

[Setting Up a New Enterprise Application](#)

[Creating a Web Application and Testing it on a Server](#)

[Modifying the Page Flow and Testing your Changes](#)

Related Topics

Once you have completed this tutorial, you may want to explore specific types of applications by reviewing other tutorials:

[Web Service](#)

[Building Enterprise JavaBeans](#)

[Accessing a Database from a Web Application](#)

[Advanced Web Services](#)

Click the arrow to navigate through the tutorial:



Step 1: Exploring the Features of Workshop for WebLogic

In this step, you will explore the features of the Workshop for WebLogic interface.

The tasks in this step are:

- [Start Workshop](#)
- [Understand the Starting Window \(Workbench\)](#)
- [Display Relevant Help Topics on the Workbench](#)
- [Display Context Help within the Workbench](#)
- [Access the Complete Online Documentation \(Including Eclipse\)](#)

A hallmark of Eclipse is providing shortcuts and multiple access points for operations. As you work through this tutorial and other Workshop for WebLogic documentation, you will find different methods to accomplish the same tasks.

To Start Workshop

If you haven't started Workshop for WebLogic yet, use these steps to do so.

... on Microsoft Windows

If you are using a Windows operating system, follow these instructions.

- From the **Start** menu, click **All Programs > BEA Products > Workshop for WebLogic Platform 9.2**

When prompted for the name of your workspace, click **OK** to accept the default workspace.

...on Linux

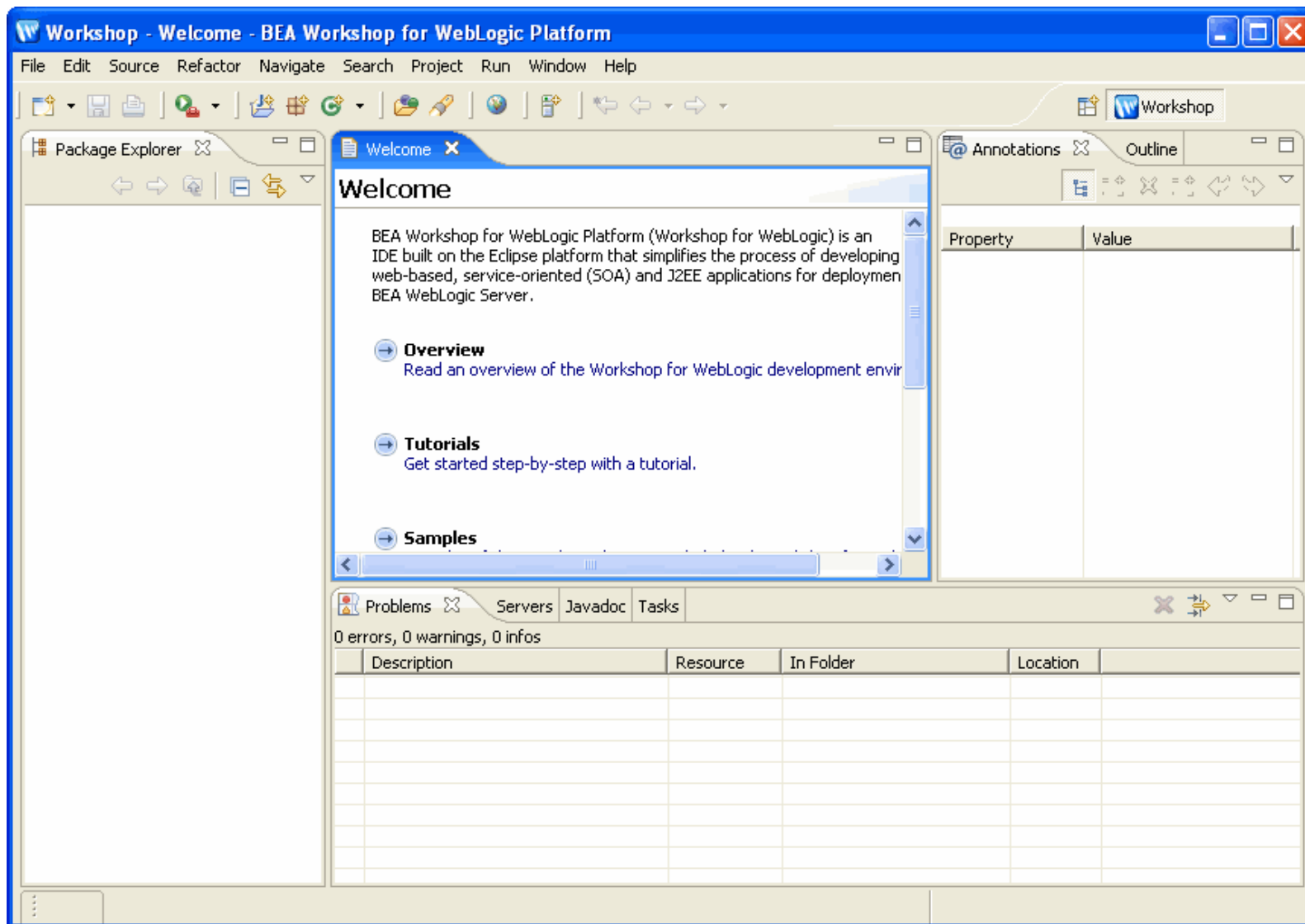
If you are using a Linux operating system, follow these instructions.

- `BEA_HOME/workshop92/workshop4WP/workshop4WP.sh`

When prompted for the name of your workspace, click **OK** to accept the default workspace.

To Understand the Starting Window (Workbench)

When you open Workshop for WebLogic, the initial window displays the workbench. The window contains a menu bar, tool bars, the editor, and information panes (called **views** in Eclipse terminology). The **Welcome** pane provides useful links to information and other resources to help you get started with Workshop for WebLogic. If you close the **Welcome** pane, you can re-display it at any time by clicking **Help > Welcome**.



The window layout is called a **perspective** and can be extensively customized. Perspectives are intended to provide related tools for performing specific tasks with specific resources. The initial perspective is called the **Workshop** perspective (shown in the upper right corner of the window). Several other useful perspectives are provided. You can switch perspectives at any time by choosing **Window > Open Perspective**.

The **Workshop** perspective is the standard perspective for developing J2EE enterprise applications with Workshop for WebLogic. Note the **Package Explorer** view at the left which allows you to move through the projects/folders/files of your workspace (currently empty).

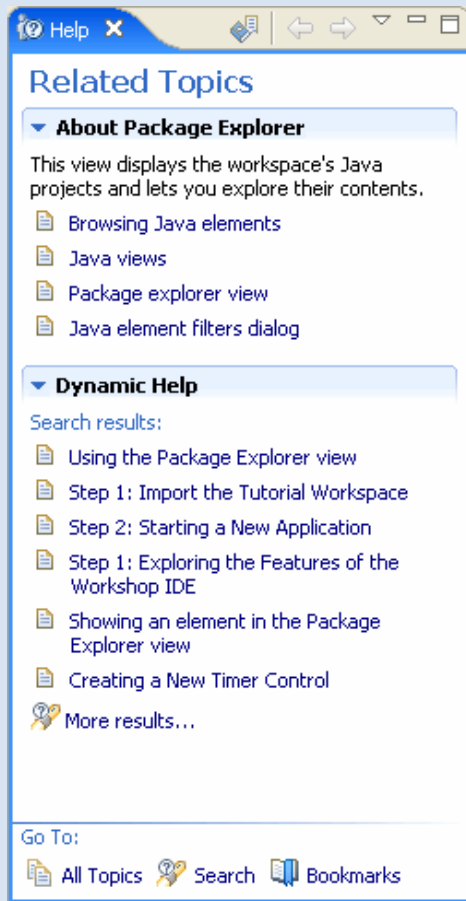
Information panes (**views**) in the workbench can be moved, torn off, displayed side by side, stacked, minimized or maximized. Each file displayed in the editor can be maximized or minimized to icons at the edge of the window (fast views). Menu bars and tool bars can be added, removed or customized. Managing the components of a perspective is described in the help system which you can access by clicking **Help > Help Contents** and choosing **Workbench User Guide**.

You can create and save your own perspectives. Workshop for WebLogic will also change the perspective when you perform other tasks. For example, a different perspective is typically used when creating a page flow or debugging.

To Display Relevant Help Topics on the Workbench

When you are getting started with Workshop for WebLogic, you may find it useful to have a list of help topics that relates to the currently selected screen element.

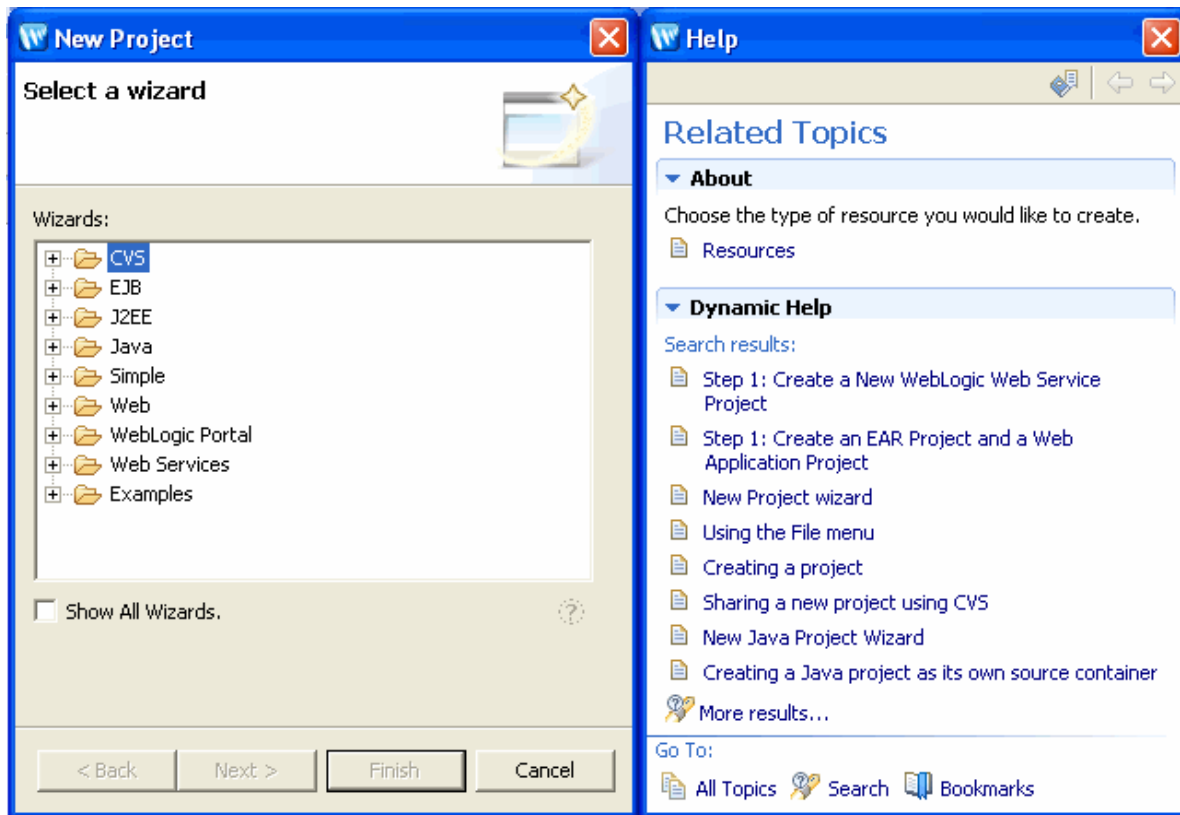
Click **Window > Show View > Other**, expand **Help** and click **Help** and **OK**. The **Help View** will be added to the workbench as shown below.



Click on different views or editor tabs on the screen and the **Help** view will update to show topics relevant to that part of the screen.

Display Context Help within the Workbench

You can also get help from dialogs. For example, if you click **File > New > Project** and then press the F1 key, a help box appears to the right of the main dialog.



To Access the Complete Online Documentation (Including Eclipse)

The workbench is described extensively in the online documentation provided on the **Help** menu.

To launch the online documentation, click **Help > Help Contents**. If you are new to Eclipse, we recommend that you browse the workbench help (including the getting started and conceptual material) before proceeding with this tutorial.

The screenshot shows a web browser window titled "Help - BEA Workshop for WebLogic Platform - Microsoft Internet Explorer". The browser's address bar and menu bar are visible. The page content is organized into a sidebar and a main content area.

Search: **GO** [Search scope:](#) All topics

Contents

- Workbench User Guide
- Java Development User Guide
- JDT Plug-in Developer Guide
- J2EE Standard Tools Developer Guide
- PDE Guide
- Platform Plug-in Developer Guide
- Web Standard Tools Developer Guide
- Web Application Development Guide
- XSD Programmer's Guide
- BEA WebLogic Portal Help
- BEA Workshop for WebLogic Platform Programmer's Guide
- Apache Beehive Documentation

BEA Workshop for WebLogic Platform Programmer's Guide

Workshop for WebLogic is an IDE built on the Eclipse platform that simplifies the process of developing web-based, service-oriented (SOA) and J2EE applications for deployment on BEA WebLogic Server.

Current Release Information:

- [What's New in 9.2](#)
- [Upgrading from 8.1](#)

Useful Links:

- [Tutorials](#)
- [Tips and Tricks](#)

Other Resources:

- [Online Docs](#)
- [Dev2Dev](#)
- [Discussion Forums](#)
- [Development Blogs](#)

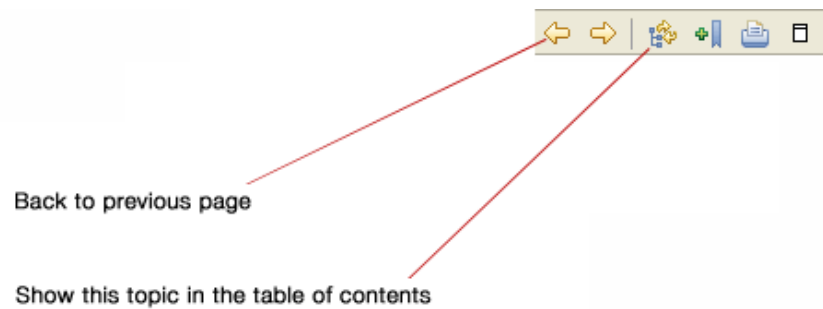
Workshop for WebLogic provides tools that automate and simplify development:

In a service-oriented application, service controls provide object-oriented access to web services, so that service operations can be accessed through simple method calls. Workshop for WebLogic handles the staging/sending/receiving/unstaging for SOAP messages that conform to the remote **web service's** service definition (WSDL).

- [Learn about these technologies](#)
- [Tutorial for developing a web service](#)
- [Developing web services in Workshop for WebLogic](#)

When developing **web applications**, Workshop for WebLogic uses the Apache Beehive NetUI framework which integrates a

Note the icons above the content pane that allow you to go back and forward (similar to standard browser back and forward buttons) and the icon through which you can see the current topic's position in the table of contents.



Note also that the help viewer is launched in a separate window, in your default web browser.

For help with Workshop for WebLogic features, choose **BEA Workshop for WebLogic Platform Programmer's Guide** from the **Contents** list.

The **Search** field at the top of the help window allows you to search for specific topics. Note that **all** online documentation (for Eclipse, Workshop for WebLogic, JDT, Eclipse plug-in development, and all available plug-ins) will be searched unless you click **Search Scope** to narrow the search to a subset of the online documentation.

Click one of the following arrows to navigate through the tutorial:



Step 2: Setting Up a New Enterprise Application

In this step, you will create the framework for a new application. The tasks in this step are:

- [Create a New Workspace for the Tutorial Application Files](#)
- [Create an Enterprise Application project](#)

Workshop for WebLogic organizes the files and components of your application in the following way:

- The top level folder is called a **workspace**.
 - Within a workspace/folder, **projects** are the top level folders, either enterprise application projects or the component files for a J2EE module; there are usually multiple projects in an enterprise application
 - **Files and folders** within a project contain project components

When starting an enterprise application, the normal process is to

1. Create a new workspace for the application
2. Create an enterprise application project to link all of the application's projects (web applications, EJBs, etc.)
3. Create projects for the modules within the application
4. Create the project contents.

Workshop for WebLogic uses the following project types:

Project Type	Project Contents
Enterprise Application (EAR) project - links component projects of the enterprise application for deployment and archiving - utility and EJB projects added to the EAR project are available on the build path of other projects in the EAR	JAR files and deployment descriptors build files and auto-generated files
Enterprise JavaBeans (EJB) project	entity beans session beans message-driven beans Java source files
Web project - web services - web applications	web services files (WSDL, Java) page flows (Java, JSP) web files (HTML, CSS, etc.) Java source files
Utility project - shared controls	Java source files

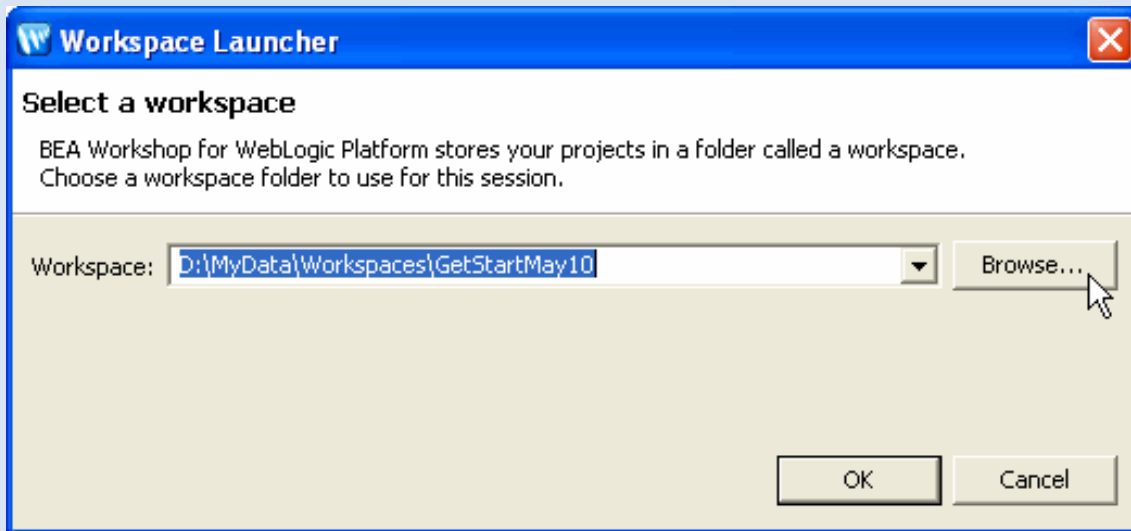
Note that the Eclipse platform provides other types of projects, but these are not enabled with the Workshop for WebLogic features.

To Create a New Workspace for the Tutorial Application

When you started Workshop for WebLogic, it prompted you for the name of your workspace. For this tutorial, you will be creating sample files and we recommend that you use a separate workspace.

To create a new workspace:

1. Click **File > Switch Workspace**.



2. In the **Workspace Launcher** dialog, click **Browse**.
3. Click **Make New Folder** and create a new folder. Click **OK**. Click **OK** to choose the new folder as your workspace.
4. WebLogic for Workshop will close the workbench window and open a new window showing the contents of the new workspace (currently empty). Be sure that you are in **Workshop** perspective. If not, click **Window > Open Perspective > Workshop** to select the correct perspective.

To Create an Enterprise Application project

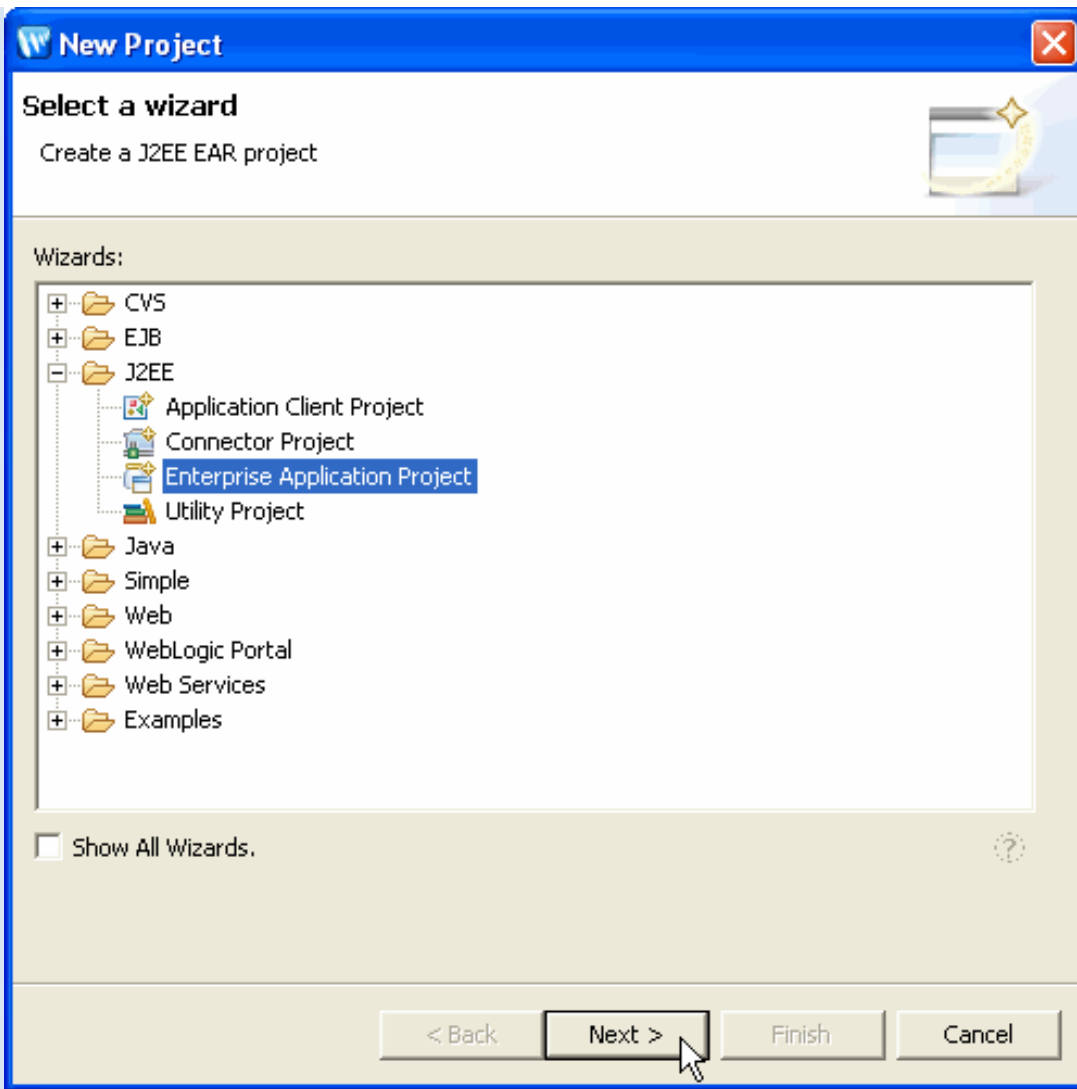
Enterprise Application projects:

- Contain JAR files that are shared by the projects in the application
- Contain links to all of the projects in the application
- Are used by Workshop for WebLogic to test/deploy enterprise applications that contain multiple projects
- Are used to create EAR (Enterprise ARchive) files

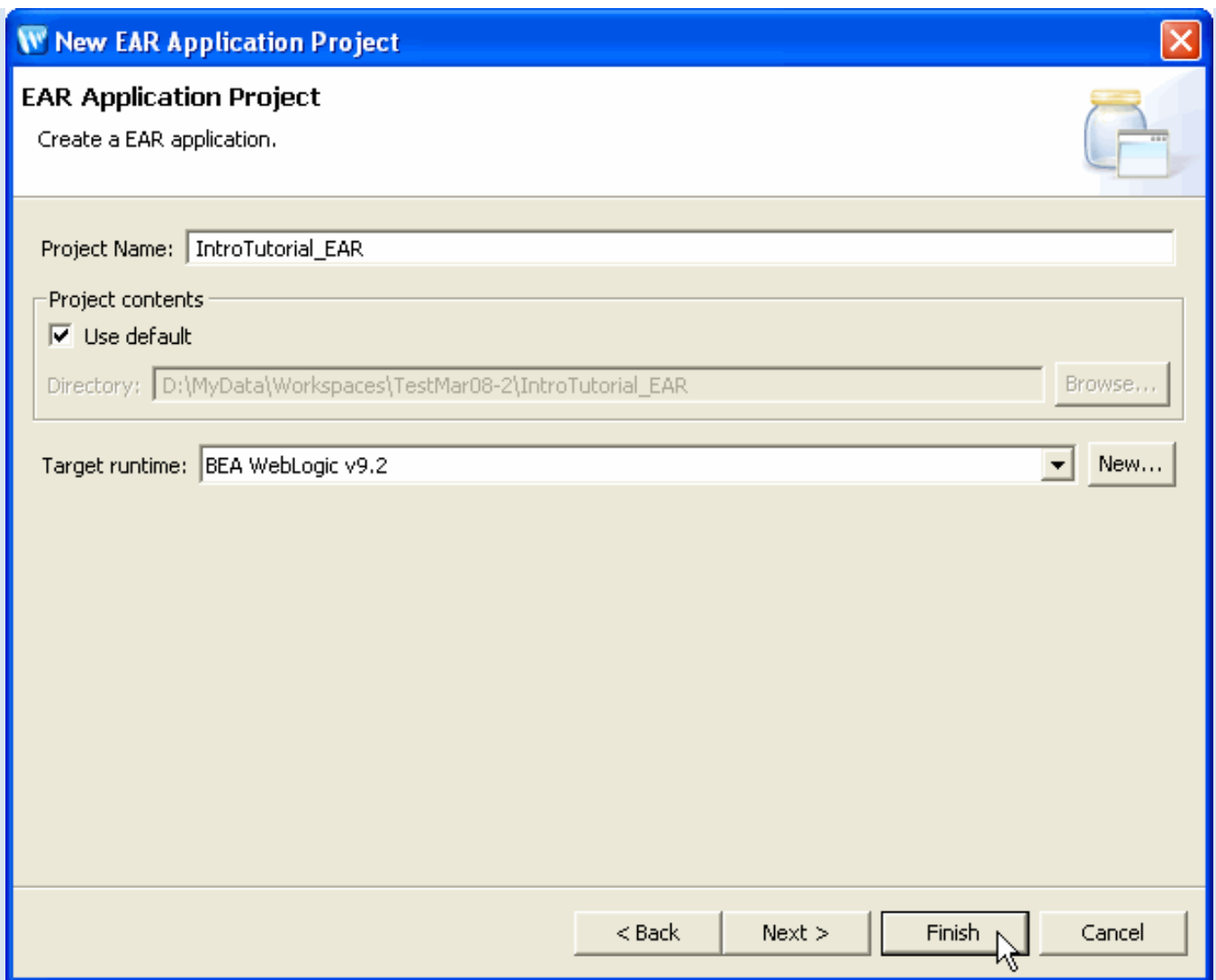
Enterprise application projects appear as siblings to the other projects in a workspace but functionally, they link together projects and do not contain any of the content of your application. A single web project (which does not contain a web service) can be deployed without an enterprise application project, but more complex applications require an enterprise application project for correct deployment. Enterprise application projects are also called EAR projects because they can be used to generate an Enterprise Archive (EAR) file for remote deployment.

To create a new enterprise application project:

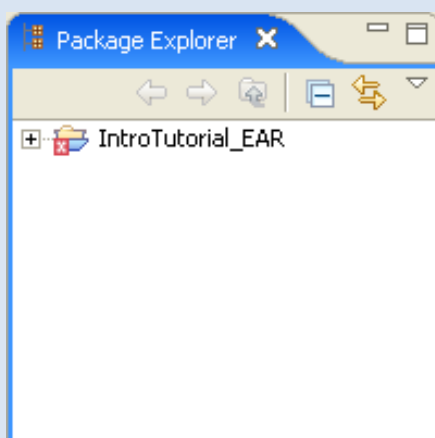
1. Click **File > New > Project...**
2. In the **New Project** dialog, expand **J2EE**, click **Enterprise Application Project**, then click **Next**.




3. In the **New EAR Application Project** dialog, in the **Project name** box, enter the name `IntroTutorial_EAR` and then click **Finish**.



4. The **Package Explorer** view displays a hierarchical list of the files and folders in the current workspace and now shows that Workshop for WebLogic has created a top-level directory for your EAR project.



Note that a **problem marker** (the red square with an X in it) may display on the folder icon  for IntroTutorial_EAR in the **Package Explorer** view.

To see a description of the problem, click on the **Problems** view at the bottom of the workbench.

Markers will appear and disappear as you create the various files in your application and fill in required components. If you

believe a marker is displayed incorrectly, try clicking **Project** > **Clean** from the menu.

Related Topics

[Applications and Projects](#)

Click one of the following arrows to navigate through the tutorial:



Step 3: Creating a Web Application and Testing it on a Server

In this step, you will create and run/deploy a complete (albeit very simple) web application.

The tasks in this step are:

- [Create a Dynamic Web Project](#)
- [Open the Initial \(Default\) Page Flow in the Web Project](#)
- [Edit the Page Flow Components](#)
- [Run the Page Flow](#)

Web applications that perform user interface functions (such as displaying "Hello, world!" in a browser page) are implemented as **page flows**. A page flow is a group of files (a Java controller class and JSP pages) that implement the flow of a portion of your user interface. When a user accesses a URL in the page flow, an instance of the controller class is created and remains in existence until the user exits the page flow. The page flow controller maintains state information for use by the controller's methods.

Web applications can contain several page flows that implement functional units. For example, one page flow could validate users, another page flow could browse data, and a third page flow might retrieve some specific information or perform navigation. Page flows in Workshop for WebLogic are based on the NetUI features of the open source framework called Beehive, a high-level project of Apache. Information on the organization and structure of NetUI and page flows is available at [NetUI Overview](#) in the Apache Beehive documentation.

The page flow controller manages state variables and the **actions** of the page flow. When the initial (default) page flow controller is created, Workshop for WebLogic inserts a class definition that looks something like this:

```
@Jpf.Controller()
public class Controller extends PageFlowController { }
```

@Jpf.Controller is an **annotation** which provides metadata for the page flow controller class. Annotations look like this:

```
@annotation-name ( parameters )
```

similar to method calls except that they begin with @. Annotations always occur immediately before the definition for the class or method to which the annotation applies--they annotate the class or method with metadata. As you modify the page flow, the annotation on the page flow class will be updated by Workshop for WebLogic.

Every page flow controller class must define the **begin** action, the entry point for the page flow. In the default class, the initial action for **begin** is defined by a parameter to the @Jpf.Controller annotation:

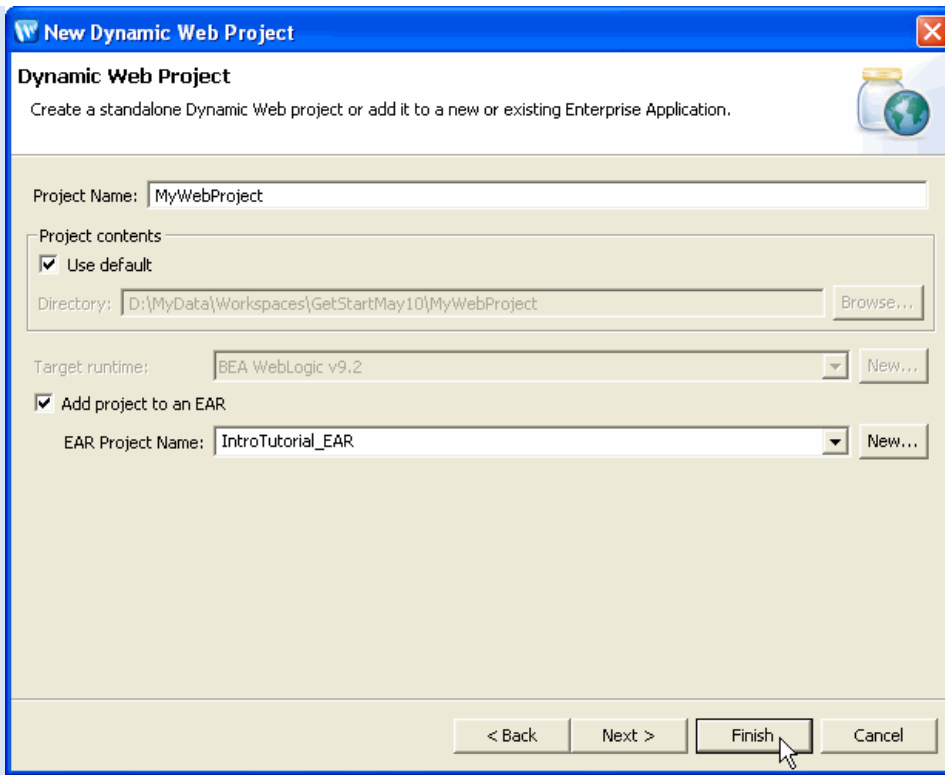
```
@Jpf.Controller(simpleActions = { @Jpf.SimpleAction(name = "begin", path = "index.jsp") })
public class Controller extends PageFlowController { }
```

The @Jpf.SimpleAction annotation specifies the **begin** action that opens the index.jsp page.

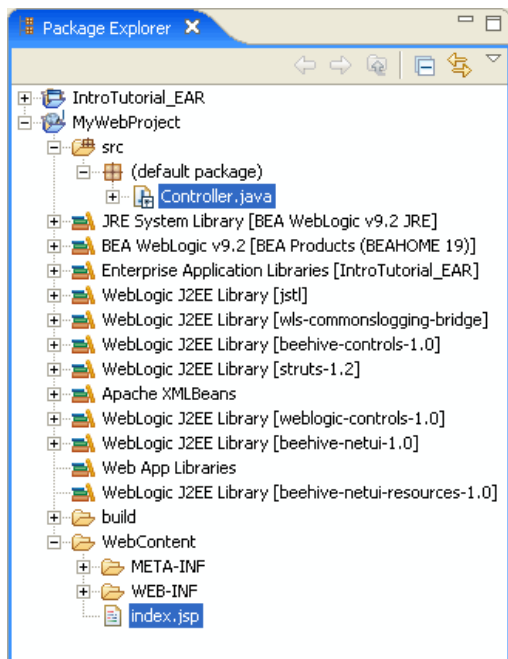
A page flow is implemented within a dynamic web project in Workshop for WebLogic. A dynamic web project can contain multiple page flows.

To Create a Web Project

1. Click **File > New > Project**.
2. Expand **Web**, click on **Dynamic Web Project**, then click **Next**.
3. In the **New Dynamic Web Project** dialog:
 - a. Enter the name `MyWebProject` in the **Project name** field.
 - b. Click on the **Add Project to an EAR** box and be sure that `IntroTutorial_EAR` is displayed in the **EAR Project Name** box.
 - c. Click **Finish**.



After you create the dynamic web project, a top level folder is created for the project and the project files are initialized. The working file set includes a default page flow consisting of two files: `Controller.java` in the `src` folder and `index.jsp` in the `WebContent` folder.

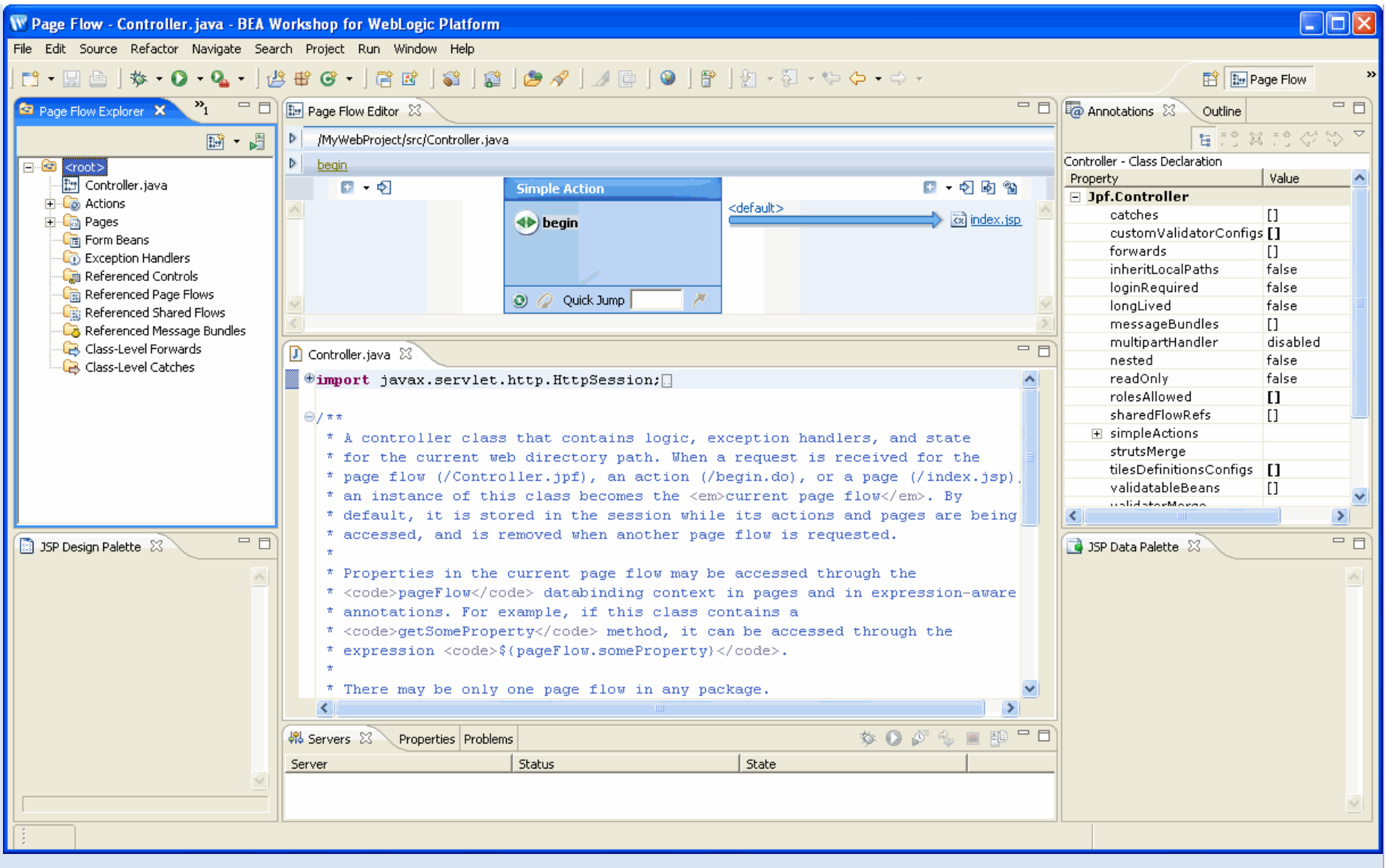


Depending on the options selected when creating the web projects, you may see a slightly different set of folders and files.

To Open the Initial (Default) Page Flow in the Web Project

To work with the default page flow controller:

1. Open the file `Controller.java` by double clicking on its name in the **Package Explorer** view. The file is located in `MyWebProject/src/(default package)`. An editor will open displaying the file contents.
2. Click **Window > Open Perspective > Page Flow**.
3. The window changes to display the **Page Flow** perspective, a layout and set of views, menus, toolbars and editor pane most useful for creating page flows. There are several points to note:
 - o A **Page Flow** tab has appeared beside the **Workshop** tab to allow you to switch perspectives.
 - o The **Page Flow Editor** is now displayed at the top of the window. The **Page Flow Editor** allows you to visually manage the links between actions and JSP pages. The initial view shows the `begin` action linked to the `index.jsp` page.
 - o The **Page Flow Explorer** view has appeared at the left. The **Page Flow Explorer** does not display the page flow files as they are organized in folders. Instead, it shows files grouped by function so that actions and pages can be accessed easily.



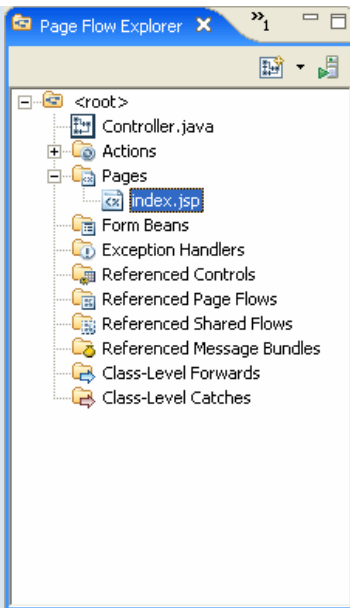
To Edit the Page Flow Components

You can display/edit other source files (JSP pages or Java files) for your page flow by double clicking on their names in the **Page Flow Explorer** view at the left. When you open a file, Workshop for WebLogic creates an **editor**-a tab in the editor area where the file is open for editing.

Double clicking an *action* opens an editor for the page flow controller (if it is already open, its tab is brought to the top of the editor area) and positions the cursor at the method for that action.

To edit the `index.jsp` file:

1. Expand **Pages** in the **Page Flow Explorer** view at the left.



2. Double click on `index.jsp`. Another editor appears in the editor area. Change the default body text to **Hello, world!** and use the **File > Save** command to save the change.

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

<netui:html>
  <head>
    <netui:base/>
  </head>
  <netui:body>
    <p>Hello, world!</p>
  </netui:body>
</netui:html>
```

To Run the Page Flow

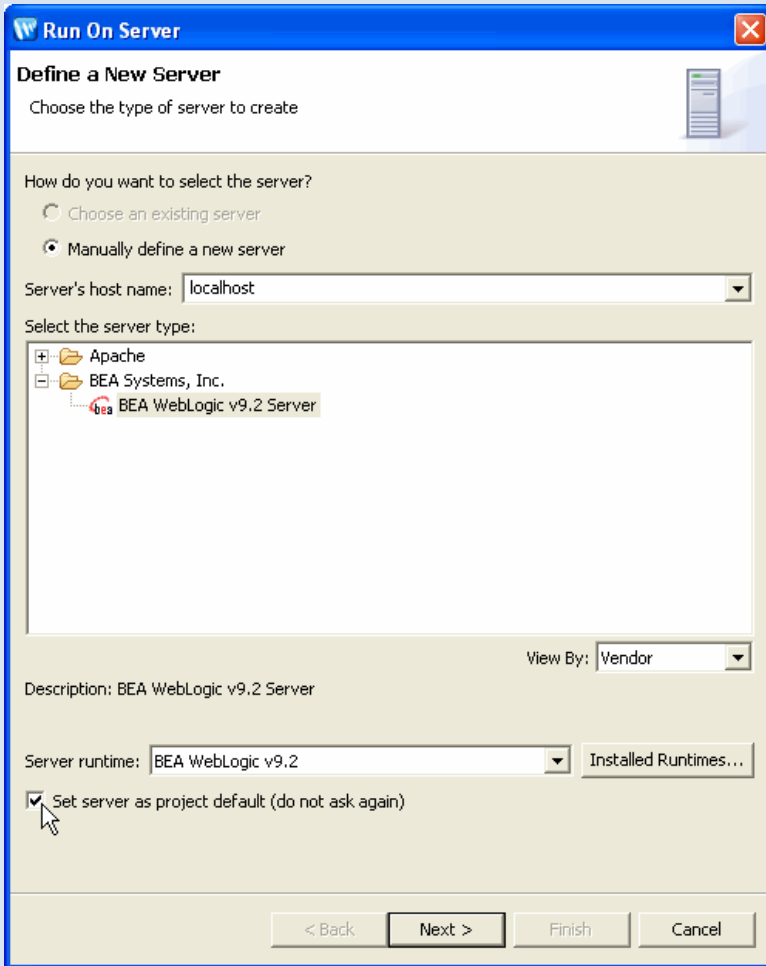
The page flow is not very interesting at this point, since all that it does is invoke the initial JSP page `index.jsp`. However, to verify that the page flow works, let's run it now.

Workshop for WebLogic creates a WebLogic Server as part of installation and we will use this local server for testing the application we will create in this tutorial. You must set up the server before you can test any applications.

If you are working through this tutorial for a second time, you must remove previous, duplicate projects (modules) from the server.

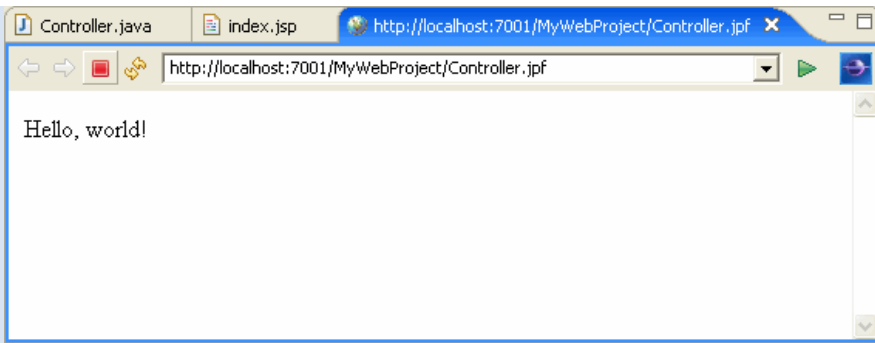
The first time you run an application, Workshop for WebLogic prompts you for information about the test server. To run your application (and set up a WebLogic Server):

1. On the editor for either `index.jsp` or `Controller.java`, right click on the editor area and from the submenu, click **Run As > Run on Server**. From the **Run on Server** dialog, click **Set server as project default** so that when you run your projects in the future you will not be prompted again. Note that **Manually define a new server** is selected since you have not yet specified the server for testing.



Click **Next** to proceed.

2. The next dialog allows you to specify the domain to use for this server. Click on the pulldown and choose `BEA_HOME\samples\domains\workshop` to use the samples domain installed with Workshop for WebLogic. For this tutorial, it is sufficient to use the sample domain shipped with Workshop for WebLogic, so click **Finish** to proceed. You may also click the link **Click here to launch Configuration Wizard to create a new domain** to [create a new domain](#).
3. Now that you have specified the server for testing, it can take a few moments while Workshop for WebLogic creates and starts the server, deploys files to the server and runs the application. While the deployment is in process, you will see status messages displayed in the **Servers** view at the bottom of the window.



The results appear in a new tab in the editor window. Note that the URL used to launch the application is `../Controller.jspf`, where the JPF (Java Page Flow) was generated from the `Controller.java` source file.

Click one of the following arrows to navigate through the tutorial:



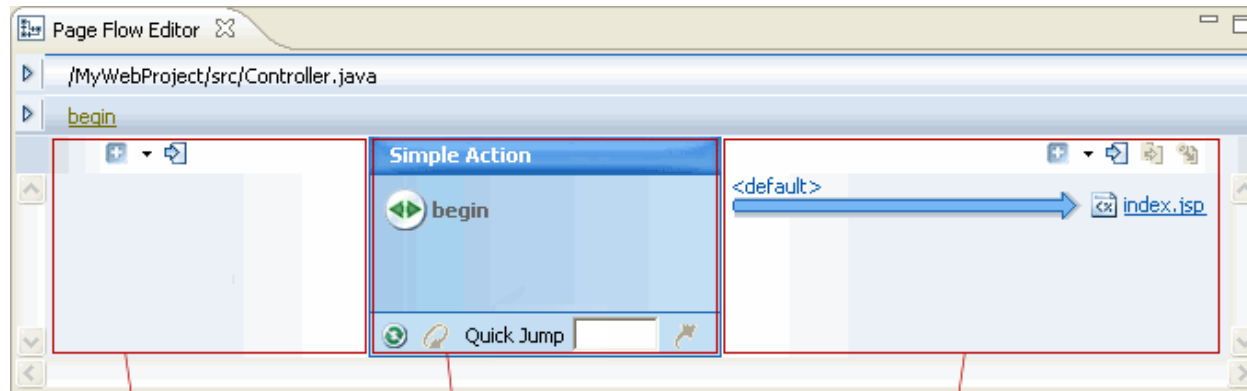
Step 4: Modifying the Page Flow and Testing your Changes

In this step, you will add a new action to the opening page (`index.jsp`) that navigates to another JSP page.

The tasks in this step are:

- [Add a new Action to the Page Flow](#)
- [Add a new Linked JSP page to the Page Flow](#)
- [Test the Expanded Page Flow](#)

The **Page Flow Editor** shows the current structure of the page flow. There are three panes on the **Page Flow Editor**:



The left pane (the upstream) shows the action or page that links **to** the action/page in the center.

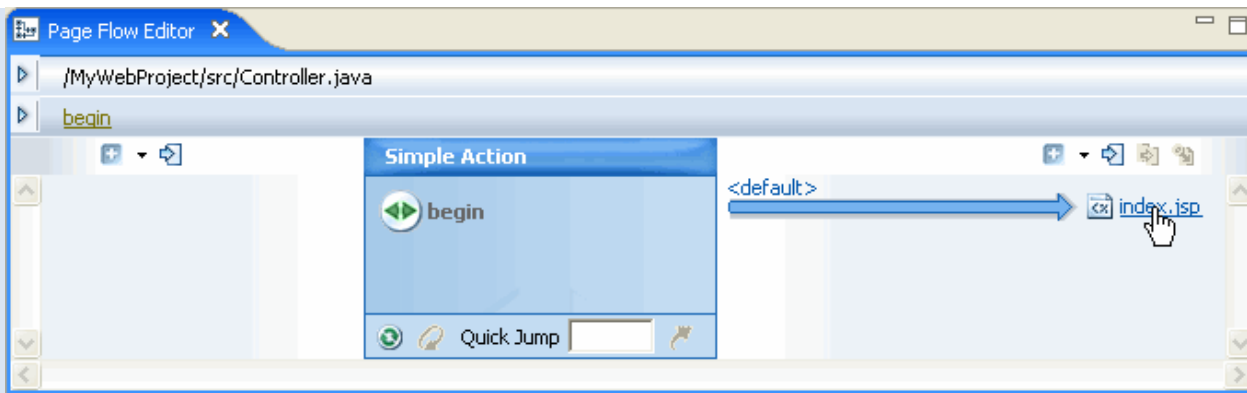
The center pane is the anchor for Page Flow Editor. The action or page displayed here is the editor's focus and everything else that takes place -- particularly with respect to editing -- happens in relation to the center pane.

The right pane (the downstream) shows the action or page linked **from** the center pane.

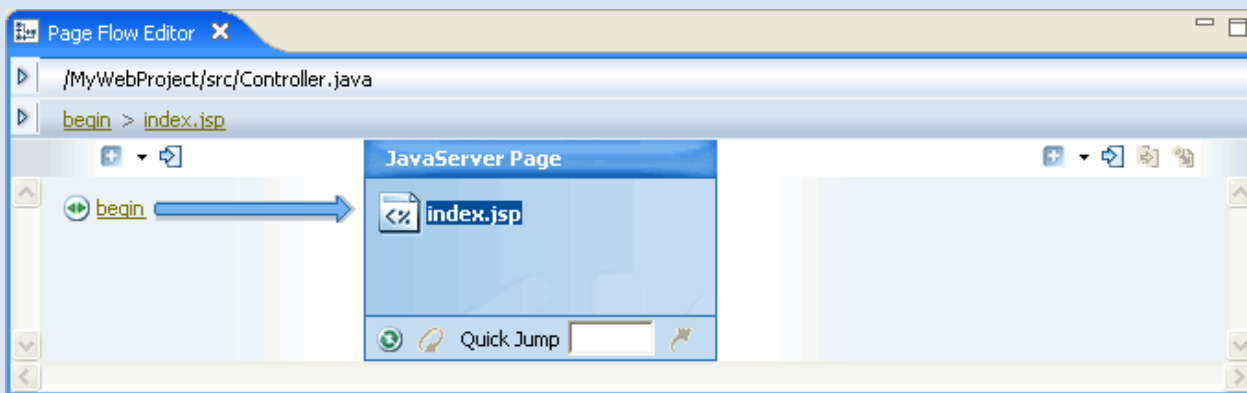
In this case, we have the simple situation of a single action, `begin`, shown in the center, and the JSP page linked to it, `index.jsp`.

To Add a new Action to the Page Flow

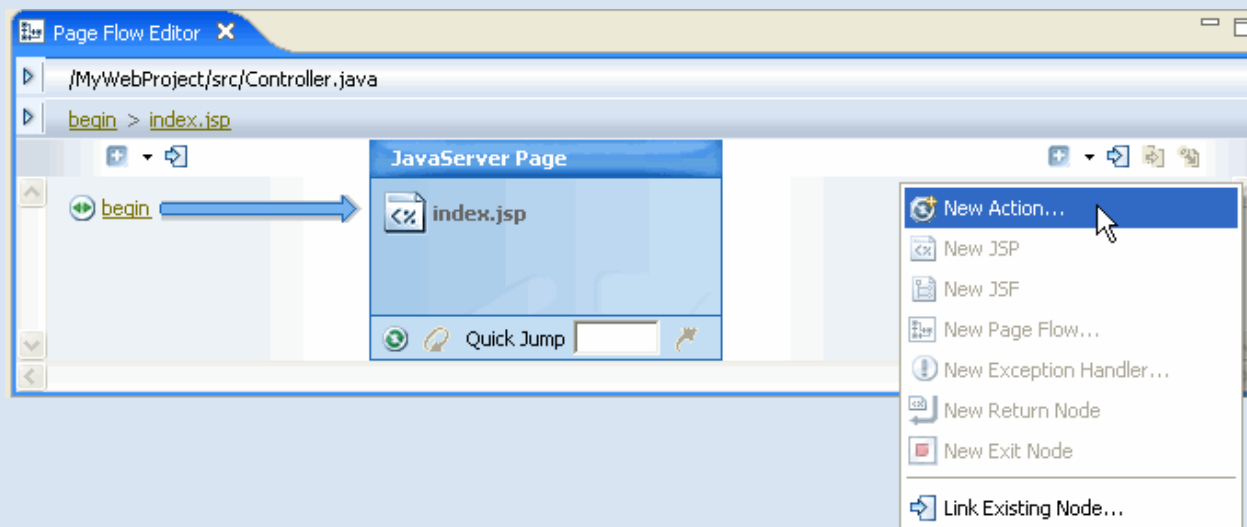
1. Click on the `index.jsp` link in the **Page Flow Editor** view.



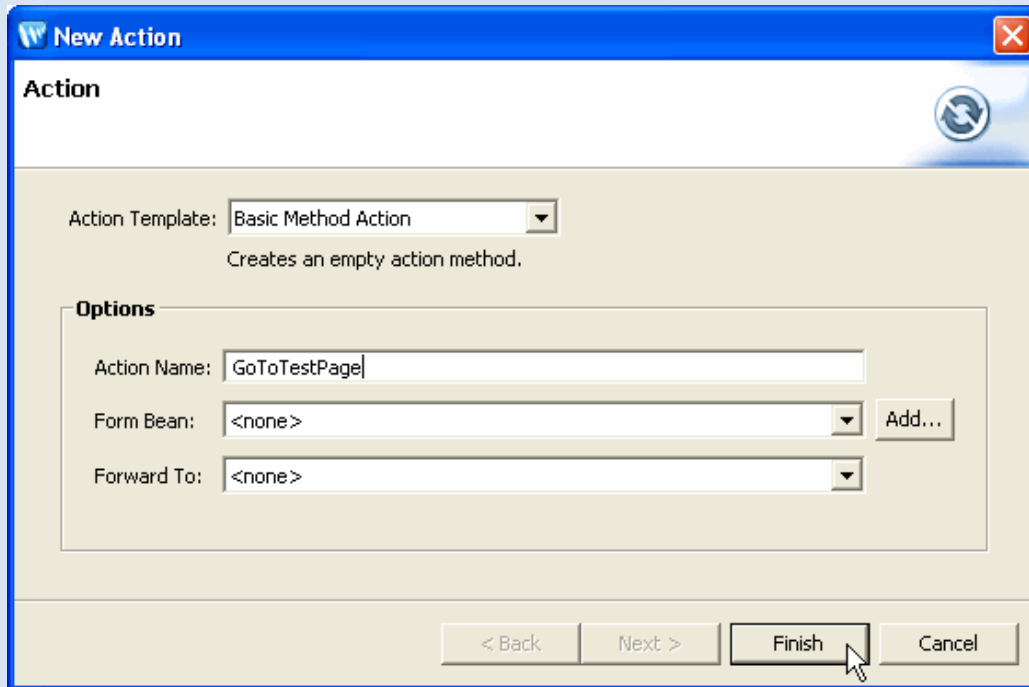
Note how this moved `index.jsp` to the center of the view.



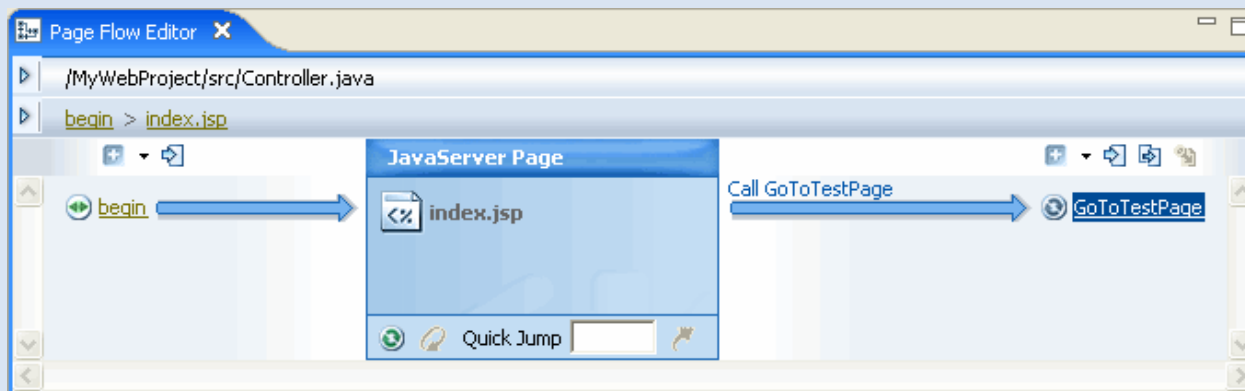
2. Right click on the downstream area (to the right) and click **New Action** from the menu.



- In the **New Action** dialog, enter the name of the new action as `GoToTestPage`. Click **Finish**.



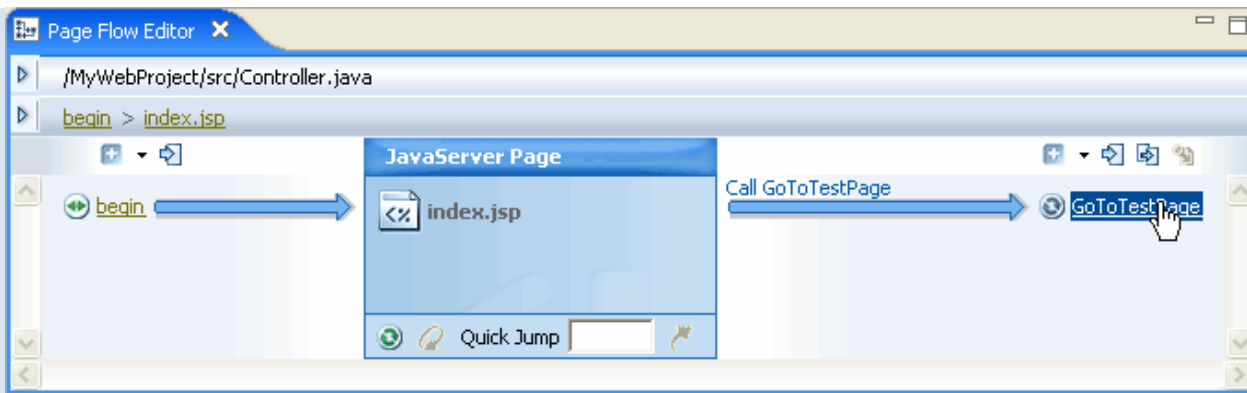
The new action appears on the downstream area of the **Page Flow Editor** and is automatically linked to the JSP page (`index.jsp`) displayed in the center of the **Page Flow Editor**.



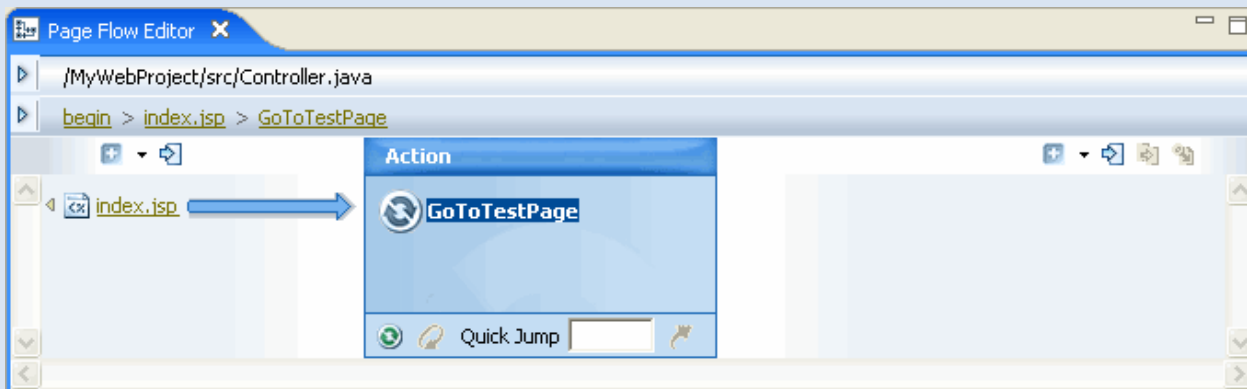
To Add a new Linked JSP File to the Page Flow

- Click on `GoToTestPage` link once.

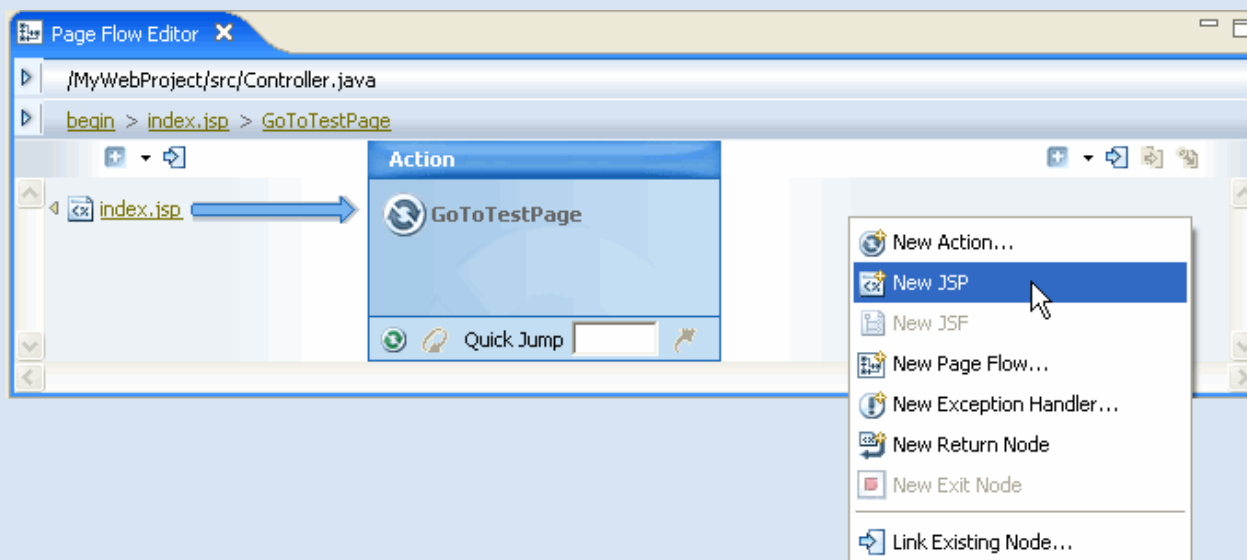
Note that clicking on the link centers the node, clicking on the icon to the left of the link simply selects the node, and double clicking on the icon goes to the corresponding source artifact.



Note how `GoToTestPage` moves into the center of the **Page Flow Editor**.



2. To create a new JSP page linked to the `GoToTestPage` action, right click on the downstream pane. Click **New JSP**.



Type the name of the new file: `TestPage.jsp` and press Enter.

- A new JSP file is created, linked to the `GoToTestPage` action. If you close the editor window displaying the result of running the original page flow, you can see the change that was made in the `index.jsp` file.

- A new JSP page has appeared in both the **Page Flow Editor** and **Page Flow Explorer**
- A link to the new page has been automatically inserted in the `index.jsp` file.

- Note that an asterisk (*) has appeared on the tab for `Controller.java` in the editor area. The asterisk indicates that the file has unsaved changes, and if you click on the `Controller.java` tab, you can see the new code that was inserted automatically when the new action was created (shown with a highlight below).

```

* There may be only one page flow in any package.
*/
@Controller(simpleActions = { @Jpf.SimpleAction(name = "begin", path = "index.js
public class Controller extends PageFlowController {
    private static final long serialVersionUID = 254128492L;

    @Jpf.Action(forwards = { @Jpf.Forward(name = "success", path = "TestPage.jsp") }
    public Forward GoToTestPage() {
        Forward forward = new Forward("success");
        return forward;
    }

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

```

- Right click on the JSP again and click **Open** to open the file in an editor tab. Change the default text to **My linked Test Page** in the body of the JSP file.

```

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

<netui:html>
  <head>
    <netui:base/>
  </head>
  <netui:body>
    <p>My linked Test Page</p>

  </netui:body>
</netui:html>

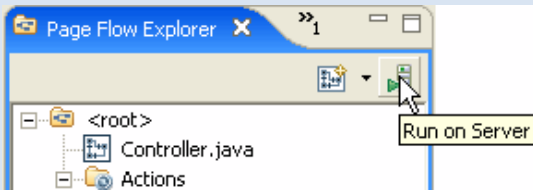
```

Click **File > Save All** to save all of the changed files in your page flow.

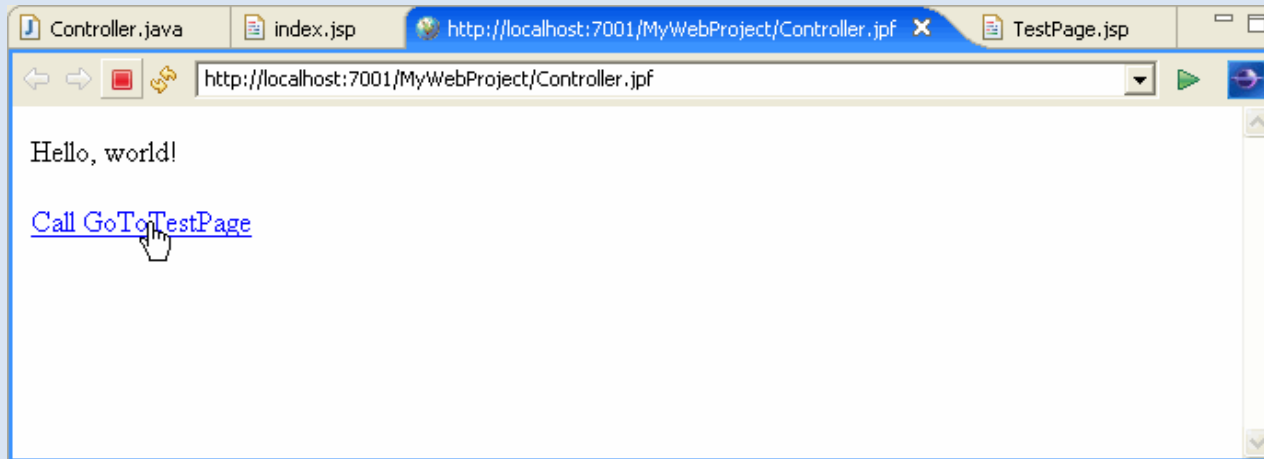
To Test the Expanded Page Flow

To run your application on the WebLogic Server:

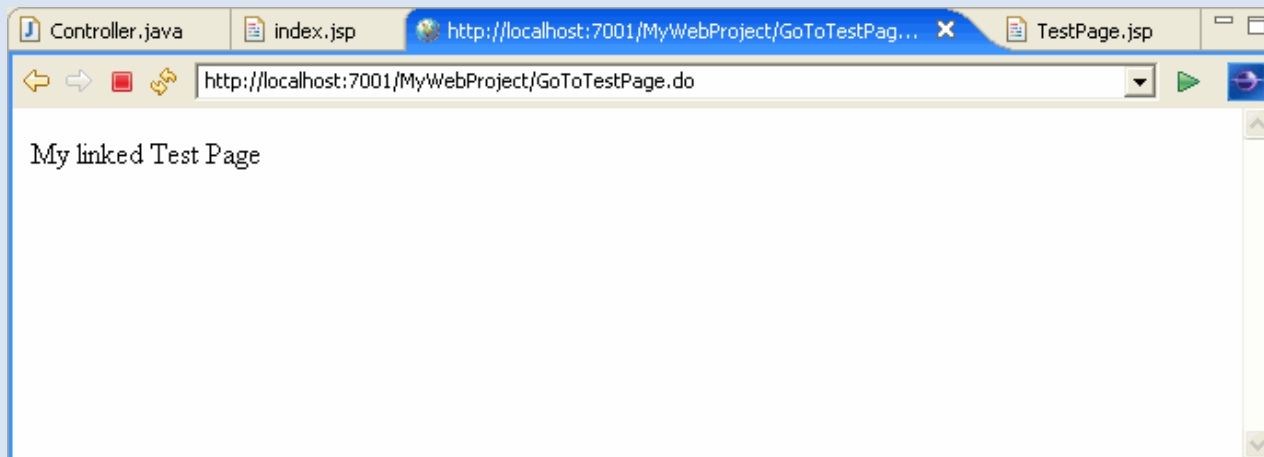
1. To run the page flow, click on the **Run on Server** button on the **Page Flow Explorer** view toolbar. If you are prompted about what tasks to perform, click **Finish** to run the application.



The page flow runs and the **begin** action is invoked, resulting in the rendering of the `index.jsp` page. The result appears in a new tab in the editor window.



2. Click on the link that invokes the `GoToNewPage` action and the second JSP page will be displayed. Displaying the second page should be very fast because at this point, the whole application has been deployed to the server and you are running your page flow.



Click one of the following arrows to navigate through the tutorial:



Summary: Getting Started with BEA Workshop for WebLogic Platform Tutorial

This topic lists the ideas this tutorial introduced, along with links to topics for more information. You may also find it useful to look at the following:

- For an introduction on using Workshop for WebLogic, see [IDE User Guide](#).
- For links to other tutorials, see [Tutorials in Workshop for WebLogic](#).
- For a list of the samples provided, see [BEA Workshop for WebLogic Samples](#).

Concepts and Tasks Introduced in This Tutorial

- Workshop for WebLogic is based on the Eclipse platform and many of the Workshop for WebLogic commands will be familiar to Eclipse users. For more information about Eclipse and its features, see <http://eclipse.org> or from the Workshop for WebLogic window, click **Help > Help Contents** and choose **Workbench User Guide**.
- An enterprise application in Workshop for WebLogic is stored in a workspace with its associated project folders and the contents of the projects.
- Workshop for WebLogic projects are typically connected into enterprise applications through a master project called an enterprise application (EAR) project. Single web projects that do not include any web services can be deployed without an EAR project.
- A page flow is a collection of files that implement a user interface function such as logging in. Workshop for WebLogic page flows are based on the Apache Beehive NetUI platform. For more information see [Web Applications](#). Beehive uses Java 5 annotations based on JSR-175, the Java specification for metadata annotations.
- Annotations specify metadata, replacing configuration files. An annotation applies to a class or method and directly precedes the class or method definition. Annotations have the form `@name (parameters)`. Annotations are created and updated as you create page flows in Workshop for WebLogic.
- To create a page flow in Workshop for WebLogic, create a web project. A default page flow is created automatically in the web project. A web project can contain multiple page flows, and you can explicitly create additional page flows in the same web project as needed.
- The initial source files for the default page flow are a controller (`Controller.java`) and a JSP page (`index.jsp`). The controller is automatically recognized as a page flow through its Beehive annotations when deployed with WebLogic Server.
- As you develop applications with Workshop for WebLogic, you test and debug code on a running instance of WebLogic Server. To test a page flow, you run it on a WebLogic Server and the results are displayed in a dynamically generated browser page in the editor area of the window.

Click the arrow to navigate back through the tutorial:



Applications and Projects

In Workshop for WebLogic you create and build **applications** (or more formally, **enterprise applications**). This topic introduces applications and their organization in Workshop for WebLogic.

Workshop for WebLogic organizes the **resources** (projects, files and folders) of your application in the following way:

- The top level folder is called a **workspace**.
 - Within a workspace, **projects** are the top level folders, either enterprise application projects or the component files for a J2EE module; there are usually multiple projects in an enterprise application
 - within a project **files and folders** contain project components

For an introduction to creating a workspace and setting up a simple enterprise application and its projects, review the [Getting Started tutorial](#). To look at sample applications, look at the [sample applications](#).

Workspaces

In Workshop for WebLogic, you can have only one workspace open at a time. A workspace is a single top level folder that contains one or more applications and component projects/modules. Workspaces are described in detail in the Eclipse help system available by clicking **Help > Help Contents** and then choosing **Workbench User Guide**.

Opening a Different Workspace

To open a new workspace, click **File > Switch Workspace** and click **Browse** to choose a new workspace folder.

Creating a New Workspace

To create a new workspace, click **File Switch Workspace** and click **Browse** to navigate to the location of the new workspace folder. Then click **Make New Folder** to create a new workspace folder.

Deleting a Workspace

A workspace is a directory. To delete a workspace and its contents, simply delete the directory on your computer.

Enterprise Applications

An enterprise application (often called simply an application) in Workshop for WebLogic is the collection of all resources that are built together and then deployed as a unit to an instance of WebLogic Server. Enterprise application components can also be archived as a unit into an Enterprise **AR**chive (EAR) file.

Enterprise applications consist of a collection of *projects* which appear as siblings in the workspace. For every enterprise application you must create the following an enterprise application (EAR) project that serves as the assembly point for all of the projects in the application.

The Java code and other files of the application are then contained in additional projects: dynamic web project(s), web service project(s), EJB project(s), and/or utility project(s).

An enterprise application is different from a *web application*. Web application refers to a collection of resources, typically HTML or JSP pages, that allow users to access functionality via the Web. An enterprise application may contain zero or more web applications.

An application should represent a related collection of business solutions. For example, if you were building an e-commerce site you might design an application that includes web applications for catalog shopping and customer account management and the components that support them (Java controls, EJBs, etc.). You might also need to deploy a human resources application for use by your employees. The human resources application is not related to the e-commerce application. Therefore, in Workshop for WebLogic you would probably create two applications to separate these disparate business functions.

When creating an enterprise application, the normal process is to

1. Create a new workspace for the application (optional)
2. Create an EAR project to link all of the application's projects
3. Create projects for the modules within the application
4. Create the project contents.

Creating a New Enterprise Application Project

To create a new enterprise application project, click **File > New > Project** and expand **J2EE**. Choose **Enterprise Application Project**. After specifying the project name and choosing the facets, the third screen of the installation wizard allows you to click **New Module** to create initial J2EE module projects. **This is not recommended** because it creates Eclipse projects, not Workshop for WebLogic projects. For a more detailed description of how to create an EAR project, see [Tutorial: Getting Started with Workshop for WebLogic Platform](#).

Component Projects

A project groups the files of a component of an application.

In Workshop for WebLogic, the available project types are:

- [Enterprise Application \(EAR\) Project](#)
- [EJB project](#)
- [Utility Project](#)
- [Dynamic Web Project](#)
- [Web Service Project](#)

Note that the Eclipse platform provides other project types. However Workshop for WebLogic features work only with the project types supported by Workshop for WebLogic.

When to Create a New Project

As you develop an application, you may need to create new projects within the application for the following reasons:

- To separate unrelated functionality

Each project should contain components that are closely related to each other. If, for example, you wanted to create one or more web services that expose order status to your customers and also one or more web services that expose inventory status to your suppliers it might make sense to organize these two sets of unrelated web services into two projects.

- To separate web services and user interface components
- To control build units

If you want to reuse controls or Java classes in multiple components, it would make sense to segregate the Java classes into a separate utility project.

Organizing Workspaces and Enterprise Application Projects

There are two common methods for organizing your projects in workspaces:

1. **All projects within the workspace.** In this structure, the project folders are created as sub-folders of the workspace folder. This organization has the advantage of keeping all application files together in one folder.
2. **Component (source) projects stored outside the workspace.** In this structure, the project folders are created outside the workspace folder. This organization has more flexibility for linking projects together, especially where you are working with multiple enterprise applications. This organization also allows for easier reassembly if the enterprise application is restructured, since the workspace folder can be recreated without affecting the project folders.

Archiving and Source Control for Project Source Code

You can archive projects and workspaces either as self-contained ZIP files or through source

control. The workspace and project folders contain metadata that should not be checked in to source control. See [Working with Source Control](#) for more information.

Summary of Project Types

Project Type	Project Contents
<u>Enterprise Application (EAR) Project</u> - links component projects of the application for deployment and archiving - one EAR project per enterprise application	JAR files and deployment descriptors build files and auto-generated files
<u>Enterprise JavaBeans (EJB) Project</u>	entity beans session beans message-driven beans Java source files
<u>Dynamic Web Project</u> - web applications	page flows (Java, JSP) web files (HTML, CSS, JPEG, etc.)
<u>Web Service Project</u> - web services	web services files (WSDL, Java) Java source files
<u>Utility Project</u> - contains shared controls and code	Java source files

EAR Project

Enterprise Application (EAR) projects:

- Contain JAR files that are shared by the projects in the enterprise application
- Contain links to all of the projects in the application
- Are used by Workshop for WebLogic to test/deploy enterprise applications that contain multiple projects
- Are used to create EAR (Enterprise ARchive) files

EAR projects appear as siblings to the other projects in a workspace but functionally, they link together projects and do not contain any of the content of your application. A single dynamic web project (web application) can be deployed without an EAR project, but more complex applications require an EAR project for correct deployment.

When an EAR project is created, by default it contains the following facets:

- EAR

- WebLogic EAR Extensions

Organization of EAR Projects

The **EarContent** directory contains the following:

EarContent	
/ APP-INF	Libraries that are available to any project in the application should be stored in EarContent/APP-INF/lib .
/ META-INF	Deployment descriptor files that may be edited by the user: <ul style="list-style-type: none"> • application.xml - lists the modules referenced by the EAR project • weblogic-application.xml - lists the modules referenced by the EAR project; these resources can be used by any module/project referenced by the EAR project

EJB Project

Enterprise Java Beans (EJBs) are Java software components of enterprise applications. The J2EE Specification defines the types and capabilities of EJBs as well as the environment (or container) in which EJBs are deployed and executed. From a software developer's point of view, there are two aspects to EJBs: first, the development and deployment of EJBs; and second, the use of existing EJBs from client software. An EJB Project provides support for creating and deploying new EJBs.

When an EJB project is created, by default it contains the following facets:

- Java
- Java Annotation Processing Support
- EJB Module
- WebLogic EJB Extensions

and can optionally have

- XMLBeans

Note that if an EJB project has dependencies on other EJB projects or on utility projects, you must create the dependency explicitly, they are not set when the project is created.

Organization of EJB Projects

The **src** directory contains the user's Java source files. Creating a package causes a new folder to be created in the **src** folder. Beans are then created within the packages.

Dynamic Web Project

Dynamic web projects are used to create web applications.

User interface components are constructed from Java Server Pages (JSPs), which are web pages that can interact with server resources to produce dynamic content. Workshop for WebLogic allows you to create page flows that define and contain the logic required to connect multiple JSPs.

A dynamic web project can contain multiple page flows.

Each dynamic web project ultimately produces a J2EE module, each of which is included in the complete Workshop for WebLogic application's EAR file when the application is built for deployment.

The contents of web projects are accessed via URLs.

When a dynamic web project is created, by default it contains the following facets:

- Dynamic Web Module
- Java
- Java Annotation Processing Support
- Beehive Controls
- Beehive NetUI
- WebLogic Web App Extensions
- WebLogic Control Extensions
- JSTL
- Struts

and may contain the following optional facets, as required:

- JSF
- Web Services
- WebDoclet (XDoclet)

- XMLBeans

Organization of Dynamic Web Projects

The **src** and **WebContent** directories contain:

src	Java source code
WebContent	Web files (e.g., JSP, graphics, HTML, CSS, etc.)
/WEB-INF	Libraries used by this web project should be stored in web/WEB-INF/lib .

Note that when you create a new page flow, a folder is created inside the **src** folder to contain the Java controller file and any other Java source files associated with the page. A folder of the same name is created inside the **web** folder to contain the JSP files associated with the page flow.

Creating a package causes a new folder to be created in the **src** folder. You can also create folders and files explicitly in the **src** directory.

Web Service Project

Web service projects are used to develop web services that conform to standards such as SOAP for message exchange, XML for messages to/from the service, and a WSDL that specifies the web service's public interface.

Web services are typically used to expose enterprise application logic to other applications (as opposed to users).

Web Services Description Language (WSDL) files are generated automatically when a web service is created within a web project.

Each web service project ultimately produces a J2EE module, each of which is included in the complete Workshop for WebLogic application's EAR file when the application is built for deployment.

The contents of web service projects are accessed via the test client which allows you to access each operation of the web service by clicking on a button.

When a web project is created, by default it contains the following facets:

- Dynamic Web Module
- Java
- Beehive Controls
- Java Annotation Processing Support
- Web Services
- Workshop Control Extensions
- WebLogic Web App Extensions

and the following facets are optional, as needed:

- Beehive NetUI
- JSF
- JSTL
- Struts
- WebDoclet (XDoclet)
- XMLBeans

Organization of Web Service Projects

The **src** and **WebContent** directories contain:

src	Java source code
WebContent	Web files (e.g., JSP, graphics, HTML, CSS, etc.)
/WEB-INF	Libraries used by this web project should be stored in web/WEB-INF/lib .

Note that when you create a new page flow, a folder is created inside the **src** folder to contain the Java controller file and any other Java source files associated with the page. A folder of the same name is created inside the **web** folder to contain the JSP files associated with the page flow.

Creating a package causes a new folder to be created in the **src** folder. You can also create folders and files explicitly in the **src** directory.

Utility Project

A utility project is a place to develop or collect general-purpose Java code that is not directly part of special entities such as web services, controls or EJBs. For example, a utility Project might hold

shared controls or Java code for a library of string formatting functions that are used in web services and JSPs in other projects in the application.

When a utility project is created, by default it contains the following facets:

- Java
- Utility Module
- Java Annotation Processing Support
- Beehive Controls
- WebLogic Utility Module Extensions
- Workshop Control Add-Ons
- XMLBeans

Organization of Utility Projects

The **src** directory contains the user's Java source files. Creating a package causes a new folder to be created in the **src** folder. You can also create folders and files directly in the **src** directory.

Note that if a utility project has dependencies on other utility projects or on EJB projects, you must create the dependency explicitly, they are not set when the project is created.

Related Topics

[Tutorial: Getting Started with Workshop for WebLogic Platform](#)

[Setting Project Facets](#)

Setting Project Facets

When a project is created, various information is assembled to specify the type of project, add standard libraries, set compiler options, control publishing tasks, set the build path and/or add an annotation processor. This information is specified by choosing **facets** during project creation. Facets can also be added and deleted from a project after its initial creation. To edit a project's facets, select **Project > Properties > Project Facets**.

Facets have version numbers. Not all facet version numbers can be changed (e.g., a facet available in only one version of software can not have other version numbers). Some facet version numbers are inter-dependent (e.g., if you choose the facet Java Annotation Processing, you must also have Java version 5.0 selected since Java versions 1.3 and 1.4 did not support annotation processing).

Each WebLogic project has two core facets:

- an enabler facet that specifies the type of project
- an extensions facet that specifies standard features that are required by this project type

For example, the Dynamic Web Project has two core facets: the Dynamic Web Module (the enable facet) and the WebLogic Web App Extensions facets.

In the project creation wizard (**File > New > Project**), required facets are displayed in **boldface**. The facet choices vary depending on the project type. For example, web service facets are not available when creating an EJB project.

WebLogic Project Facet Types

Enabler Facets

The enabler facets specify the WebLogic project type. The following table lists the available enabler modules.

Facet Name	Version Info	Description
EAR	J2EE spec version (1.4, 1.3, 1.2)	Enables the project as an EAR
EJB Module	EJB spec version (2.1, 2.0, 1.1)	Enables the project for EJBs
Utility Module	WTP version for the utility module facet (1.0)	Enables the project to be referenced by other projects (J2EE modules)
Dynamic Web Module	Servlet spec version (2.4, 2.3, 2.2)	Enables web applications (web services and page flows)

Minimum Project Extensions

All WebLogic projects require, as a minimum, support for J2EE shared libraries. This WebLogic feature allows modules to share a single copy of the J2EE libraries rather than duplicating the library in each project. The following table lists the available project extensions.

Facet Name	Version Info	Adds (in addition to J2EE shared library support):
WebLogic EAR Extensions	WebLogic Server version (9.2)	
WebLogic EJB Extensions	EJB spec version (2.1, 2.0, 1.1)	EJBGen tool that allows EJBs to be created in a single annotated .java source file
WebLogic Utility Module Extensions	WebLogic Server version (9.2)	
WebLogic Web App Extensions	WebLogic Server version (9.2)	

Java Support

Java language support and annotation processing are required by all working projects (web applications, EJBs, utility projects). EAR projects require only Java language support.

Facet Name	Version Info	Description
Java	Java version number (5.0, 1.4, 1.3)	Add Java edit, refactor, compile features (the JDT tools)
Java Annotation Processing Support	Java version number (5.0)	Support for Java 5.0 annotation (JSR-175)

XMLBeans Support

XMLBeans is a project of the Apache Foundation (<http://xmlbeans.apache.org>). The XMLBeans feature of WebLogic can be used with any project type and replaces the schema project type used in WebLogic Workshop 8.1. The XMLBeans builder compiles complex data types in WSDLs or schemas into Java types. This is used when building a web service with an existing schema. It is also useful when accessing a web service control that incorporates complex types. For more information on using XMLBeans, see [Using XMLBeans in the IDE](#).

XMLBeans are optional within all projects types.

Facet Name	Version Info	Description
XMLBeans Library	XMLBeans version 2.0	Puts XML Beans API on the build path
XMLBeans Builder	XMLBeans version 2.0	Adds builder plugin to Eclipse that generates Java types from WSDL or XSD (schema) files.

XDoclet support

XDoclet is an extended doclet code generator for Java. XDoclet is a tool that allows developers to create servlets and EJBs in a single annotated source file and XDoclet then automatically generates XML descriptors and interfaces.

Facet Name	Version Info	Description
EJBDoclet (XDoclet)	XDoclet spec version number (1.2.3, 1.2.2, 1.2.1)	Enables the project to run EJBDoclet post-processing on annotated EJBs
WebDoclet (XDoclet)	XDoclet spec version number (1.2.3, 1.2.2, 1.2.1)	Enables the project to run WEBDoclet post-processing on annotated servlets

Beehive Page Flows and Controls

Beehive is a project of the Apache Foundation (<http://beehive.apache.org>). Beehive provides support for JSR-175 annotations and includes NetUI and controls.

Facet Name	Version Info	Description
Beehive Controls	Beehive version (1.0.1)	Adds Beehive system controls and support for custom controls
Beehive NetUI	Beehive version (1.0.1)	Adds support for page flows and JSPs as well as integration with Java Server Faces (JSF) and Struts
Workshop Control Extensions	WebLogic Server version (9.2)	Workshop for WebLogic extensions for controls including timer control and web service control

JSF (Java Server Faces)

Java Server Faces is a component framework for building user interfaces for web apps

Facet Name	Version Info	Description
JSF	Sun Reference Implementation 1.1.01	Adds JSF implementation.

Annotated Web Services

JSR-181 defines a standard annotated Java format that uses Java Language Metadata (JSR-175) to enable easy definition of Java Web Services in a J2EE container.

Facet Name	Version Info	Description
Standard Annotated Web Services	JSR-181 version (1.0)	Adds support for standard annotation as defined in JSR-181
WebLogic Web Service Extensions	WebLogic Server version (9.2)	Adds WebLogic extensions to standard JSR-181 annotations

J2EE Support

Additional facets provide standard J2EE libraries and APIs.

Facet Name	Version Info	Description
JSF	JSF version	JavaServer Faces
Struts	Struts version	Struts; this is required by Beehive NetUI
JSTL	JSTL version	JSP standard tag library

Related Topics

[Web Application Technologies](#)

[Applications and Projects](#)

Managing Project Dependencies

A *dependency* is a situation where a project requires some other component to build or deploy correctly. This topic describes how to set up and manage dependencies between components in order to assemble a working enterprise application. For the most part, these dependencies are created automatically for you, and the information in this topic is only required if you wish to modify or delete dependencies.

Understanding Project Dependencies

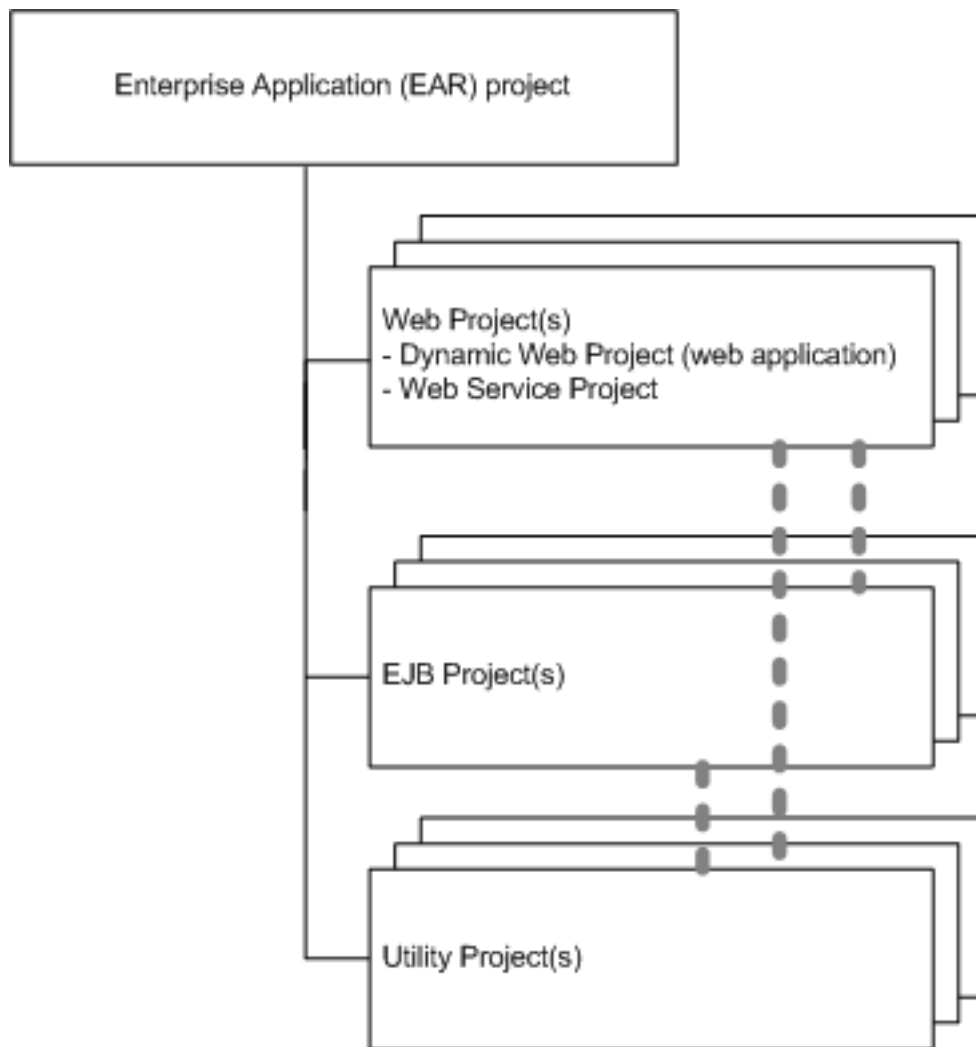
The assembly point for an enterprise application is an enterprise application (EAR) project. Other projects are added to the EAR project as necessary:

- Web services projects
- Dynamic web projects (web applications)

and shared projects are added to the EAR and made available to the existing web projects:

- Utility projects (shared code that is used by the web projects)
- EJB projects (J2EE EJBs that are used by the web projects)

The usual structure looks something like this where all projects are connected to the application (as shown by solid lines in the diagram below), and the shared projects are also made available to the web projects (the thick dotted lines in the diagram):



To build an enterprise application correctly, all shared projects must be correctly connected to the web projects that depend on the shared code.

To deploy an enterprise application correctly, all web services, dynamic web projects (web applications), utility and EJB projects must be added to the EAR project. All projects in an EAR are deployed together, whether to a server or for archiving to an EAR file.

Dependencies on Other Code Sources

In addition to project dependencies, projects may also require:

- JAR files
- System J2EE libraries (runtime modules that are normally specified through facet selections, and that are required to run the enterprise application)
- J2EE libraries that are not part of the standard Workshop for WebLogic runtime library set

Topics Included in This Section

Configuring Standard Project Dependencies

Describes how to add projects to an EAR and how to ensure that utility/EJB projects are

added to web projects.

Setting up JAR and J2EE Library Dependencies

Discusses how to manage JAR files and WebLogic J2EE shared libraries.

Customizing Project Dependencies

Describes methods for managing project dependencies more precisely.

Related Topics

none.

Configuring Standard Project Dependencies

There are three kinds of project dependencies:

1. All projects (web services, dynamic web, utility and EJB projects) must be added to an EAR project in order to deploy together either to a server or to an archive. **Can be set automatically at project creation.**
2. If a utility or EJB project is added to an EAR that already contains a web services project(s) or dynamic web project(s), the new utility/EJB project must be made available to each web project (web service project and dynamic web project) to ensure that the utility/EJB project is available to the web project at build time. **Can be set automatically at project creation.**
3. If utility or EJB projects have dependencies on other utility or EJB projects, the projects must have those dependencies set explicitly. Note that you must not create circular dependencies.

The normal process creates dependencies automatically if you:

- Create an EAR project first and then as new projects are created, click the **Add project to an EAR** option in the project creation wizard to add the projects to the EAR

OR

- Create your projects and then create a EAR project and from the project creation wizard, add projects to the EAR.

You may need to configure these dependencies explicitly if:

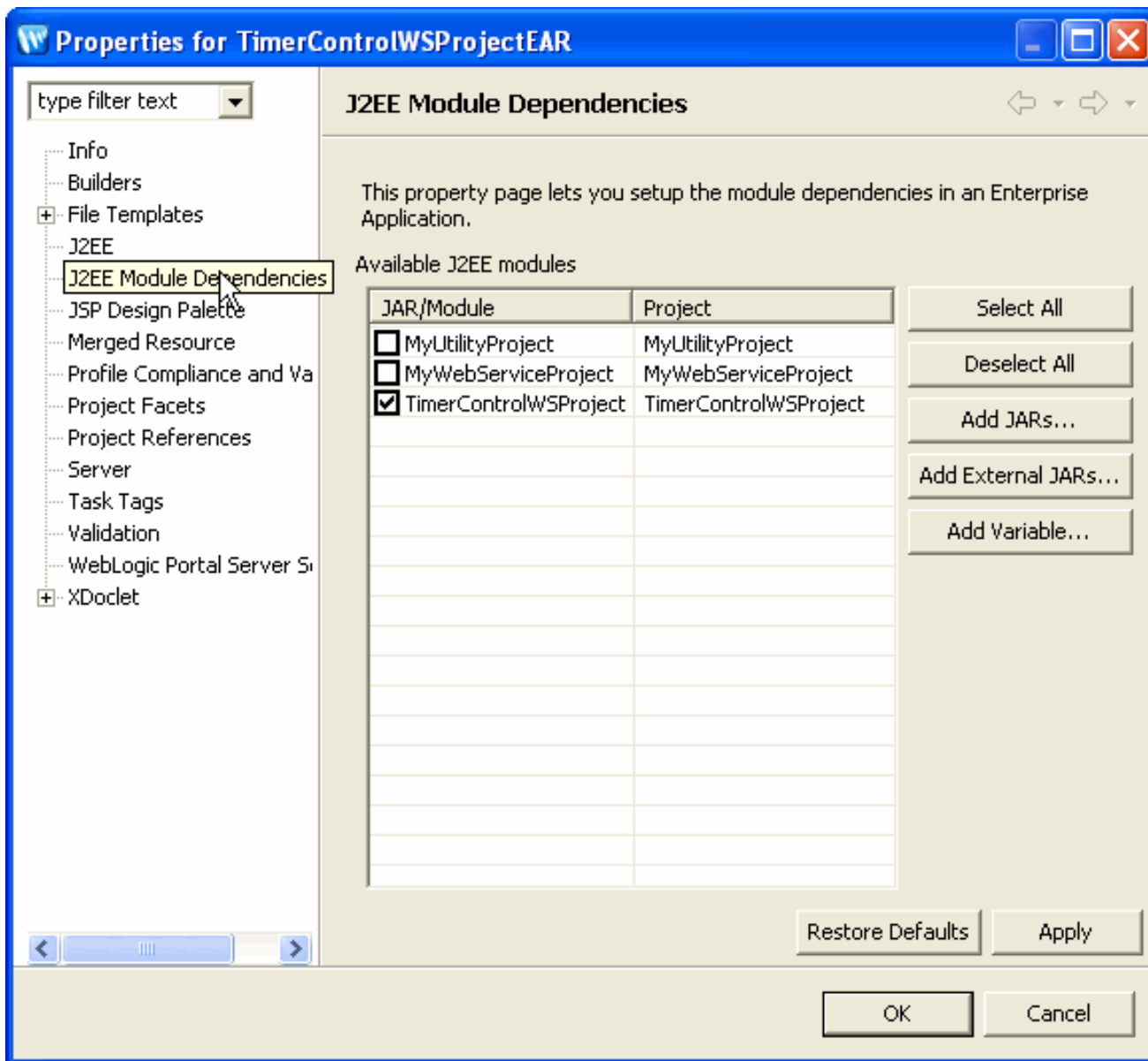
- A project was imported (rather than created) in your workspace
- A project was not added to the EAR at creation time
- Projects do not compile correctly because a utility or EJB project is not available at build/run time
- The IDE does not recognize references to the utility project or EJB project objects when editing source files

Step 1: Adding Projects to an EAR

You may need to configure this dependency explicitly if projects compile correctly locally but do not deploy correctly to the server because not all projects are in the EAR. All of the projects associated with an EAR must be added to the EAR, including EJB projects, utility projects, dynamic web projects and web service projects.

To add a project to an EAR project:

1. Right click on the EAR project name in the **Package Explorer** view and choose **Properties** from the context menu. From the **Properties** dialog, click on **J2EE Module Dependencies** in the bar at the left. In the sample below, there are three projects, one already added to the ear and two that are not added.



2. Click in the box beside the name of each project to be added to the EAR project. Click **OK** to add the projects to the EAR.

Note that adding a utility or EJB project to an EAR does not make the project available to web projects in the same EAR. To add a utility or EJB project to the web projects in the EAR, follow the process described below for every project in the EAR that has a dependency on that project.

Step 2: Adding Utility and EJB Projects to Web Projects

Since EJB and utility projects contain code that is used by web service projects or dynamic web projects, the usual option is to have EJB and utility projects available to all projects within an EAR. In practical terms, this means that

- Utility projects are compiled into libraries that are stored in the APP-INF/lib folder of the EAR, to be shared by all projects in the EAR. These libraries are then inserted on the build path for each of the web projects.
- EJBs are loaded at runtime and available for all web projects/modules in the enterprise application.

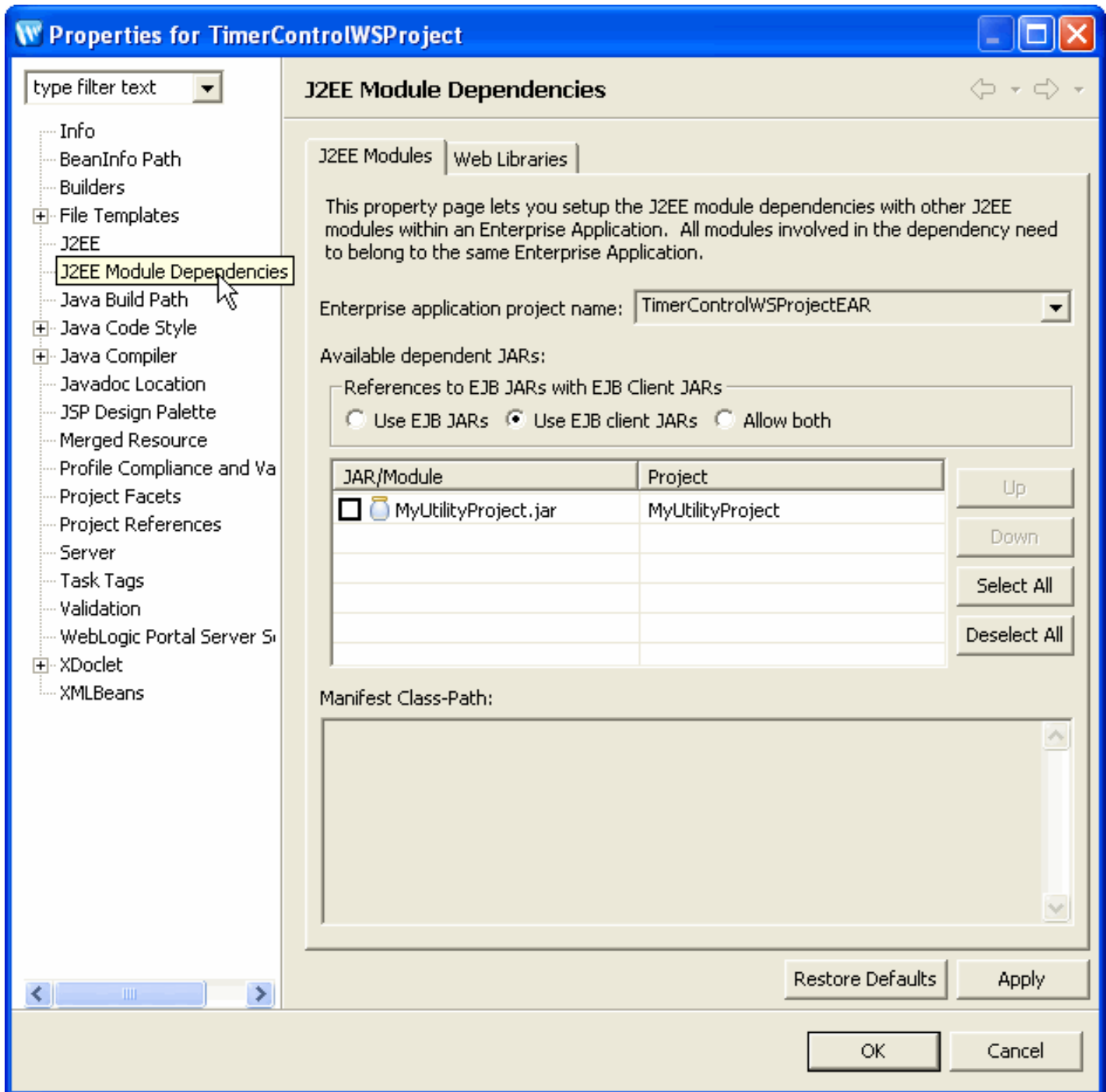
However utility and EJB projects are not available at build time unless the project dependencies are configured correctly.

To specify that utility and EJB projects are available to all projects in an EAR, the simplest method is to check the **Add**

dependencies from all WebLogic Web modules in EAR to this module checkbox on the third screen of the creation wizard (the default setting) when creating a utility or EJB project. This is the default setting.

If the project dependencies for utility/EJB projects are not set at project creation, you can make a utility or EJB project available to the web services project and dynamic web projects in your EAR. For each web project:

1. Right click on the web services or dynamic web project name in the **Package Explorer** view and choose **Properties** from the popup menu. From the **Properties** dialog box, click on **J2EE Module Dependencies** in the bar at the left. Be sure that the **J2EE Modules** tab is selected. In the sample below, there is a new utility project which is not added to the current project.



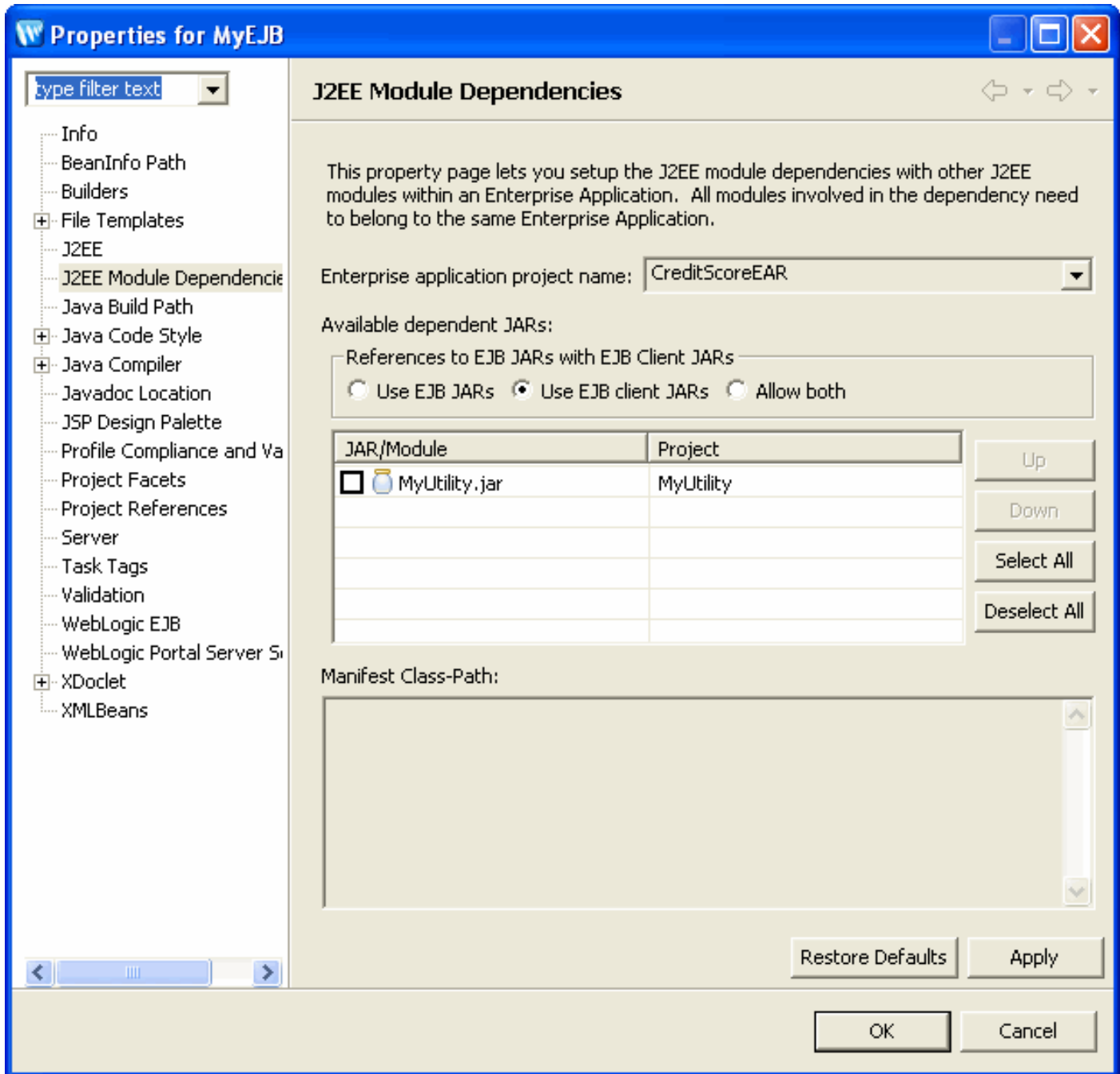
2. Click in the box beside the name of each project to be added. Click **OK** to add the utility or EJB project(s) to the web project.

Note that the **References to EJB JARs with EJB Client JARs** box has no effect in Workshop for WebLogic projects.

Step 3: Making Utility and EJB Projects Available to Other Utility/EJB Projects

Utility and EJB projects may have dependencies on each other. If this is the case,

1. Right click on the utility or EJB project name in the **Package Explorer** view and choose **Properties** from the popup menu. From the **Properties** dialog box, click on **J2EE Module Dependencies** in the bar at the left. Be sure that the **J2EE Modules** tab is selected. In the sample below, there is a utility project which is not added to the current project.



Click in the box beside the the name of each project to be added. Click **OK** to add the utility or EJB project(s) to the current utility or EJB project.

Note that the **References to EJB JARs with EJB Client JARs** box has no effect in Workshop for WebLogic projects.

Note that you should not create circular dependencies among EJB and utility projects.

Related Topics

[Applications and Projects](#)

[Understanding the Build Process](#)

[Deploying an Application for Testing](#)

Setting up JAR and J2EE Library Dependencies

This topic describes techniques for managing the location and availability of JARs and WebLogic J2EE Shared Libraries in your projects and enterprise applications.

The IDE builds projects using project source files as well as JARs that are stored in the project, or available on the build path. Within the IDE, virtual copies of J2EE modules may be stored in the project, rather than physical copies. After build, the resulting module is deployed to the server. Modules may access runtime libraries available on the class path.

There are two types of libraries that you can add to a project or application

- JAR files (plain old Java files assembled into a JAR file; normally simply copied into APP-INF/lib or WEB-INF/lib) - these are also called **Web Libraries** in Eclipse/WTP terminology
- J2EE libraries (also called shared J2EE libraries) are JAR files that can be deployed once to the server and then accessed by many modules or applications.

For more information on J2EE libraries, consult your WebLogic Server documentation locally or [online](#). J2EE libraries can be:

- System (runtime) libraries. These J2EE libraries are added when you add facets to your project, either at creation time or from the **Project Facets** properties page.
- Custom libraries that you create and register as shared libraries with **Window > Preferences > WebLogic > J2EE Libraries**. These must be added individually. Consult the WebLogic Server documentation for more information about J2EE libraries.

You can:

- [Add JARs to your projects](#)
- [Manage How Facets Make J2EE Libraries Available to Projects](#)
- [Make Custom J2EE Libraries Available to your Projects](#)

Adding JARs to Your Projects

You can add JARs to your projects by copying them to your project folders. Note that you may need to right click on the **Package Explorer** view and click **Refresh** to update the file list.

Option 1: Add a JAR to an Enterprise Application

JAR files copied into the **EarContent/APP-INF/lib** directory of an EAR project are added to the buildpath of all projects included in the EAR. You do not need to configure project dependencies on such JAR files. Objects in the JAR will simply appear in the IDE and they will be available at build and deployment for all projects in the EAR.

Option 2: Add a JAR to a Single Project

JAR files copied into the **WebContent/WEB-INF/lib** directory of a web project are added to the buildpath of that project. The contents of the JAR file will appear in the IDE and they will be available at build and

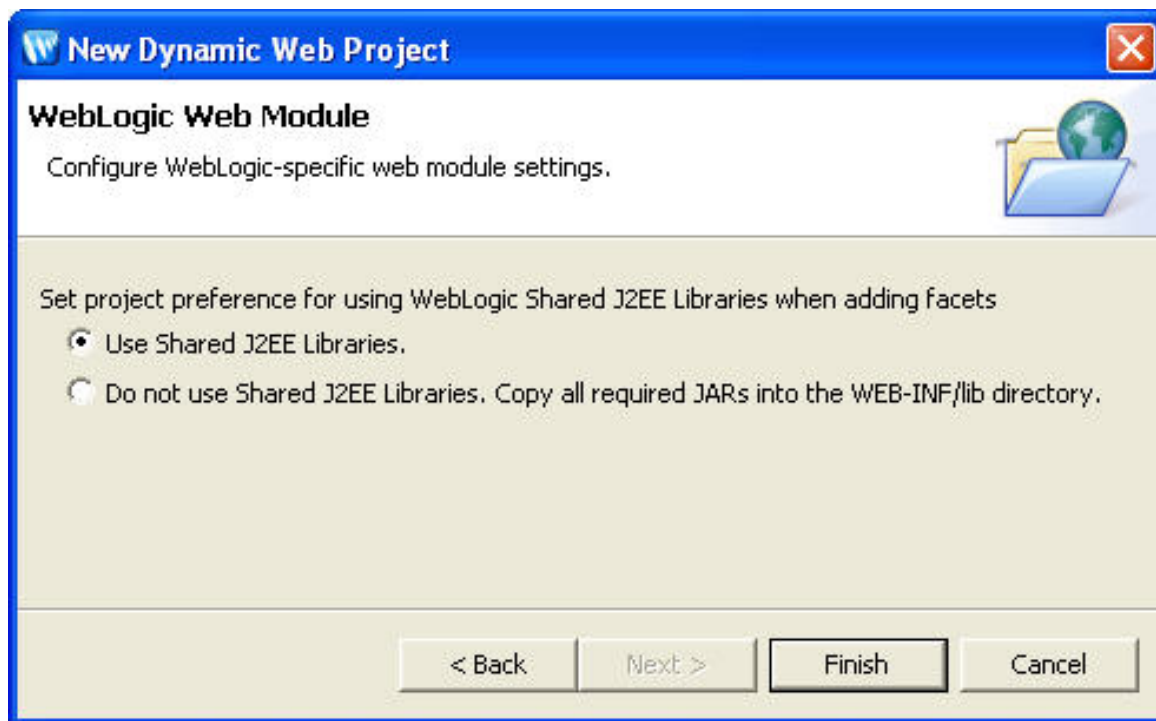
deployment for that single project only.

Managing How Facets Make J2EE Libraries Available to Projects

When creating a project in Workshop for WebLogic, the project facets that are selected for the project determine what J2EE libraries are required. You may also add or remove project facets after a project is created. J2EE libraries can be handled one of two ways:

1. Workshop for WebLogic allows you to use the WebLogic Server shared J2EE libraries feature which makes the J2EE libraries available on the build path but does not make copies of the J2EE libraries in every project where it is used.
2. The J2EE standard method is to copy the JARs inside a J2EE library into the WEB-INF/lib folder of the project.

When you create a web project, one of the screens allows you to specify whether this project will share J2EE libraries (**Use Shared J2EE Libraries**) or create a copy of the J2EE libraries in the current project (**Do not use Shared J2EE Libraries. Copy all required JARs into the WEB-INF/lib directory**).

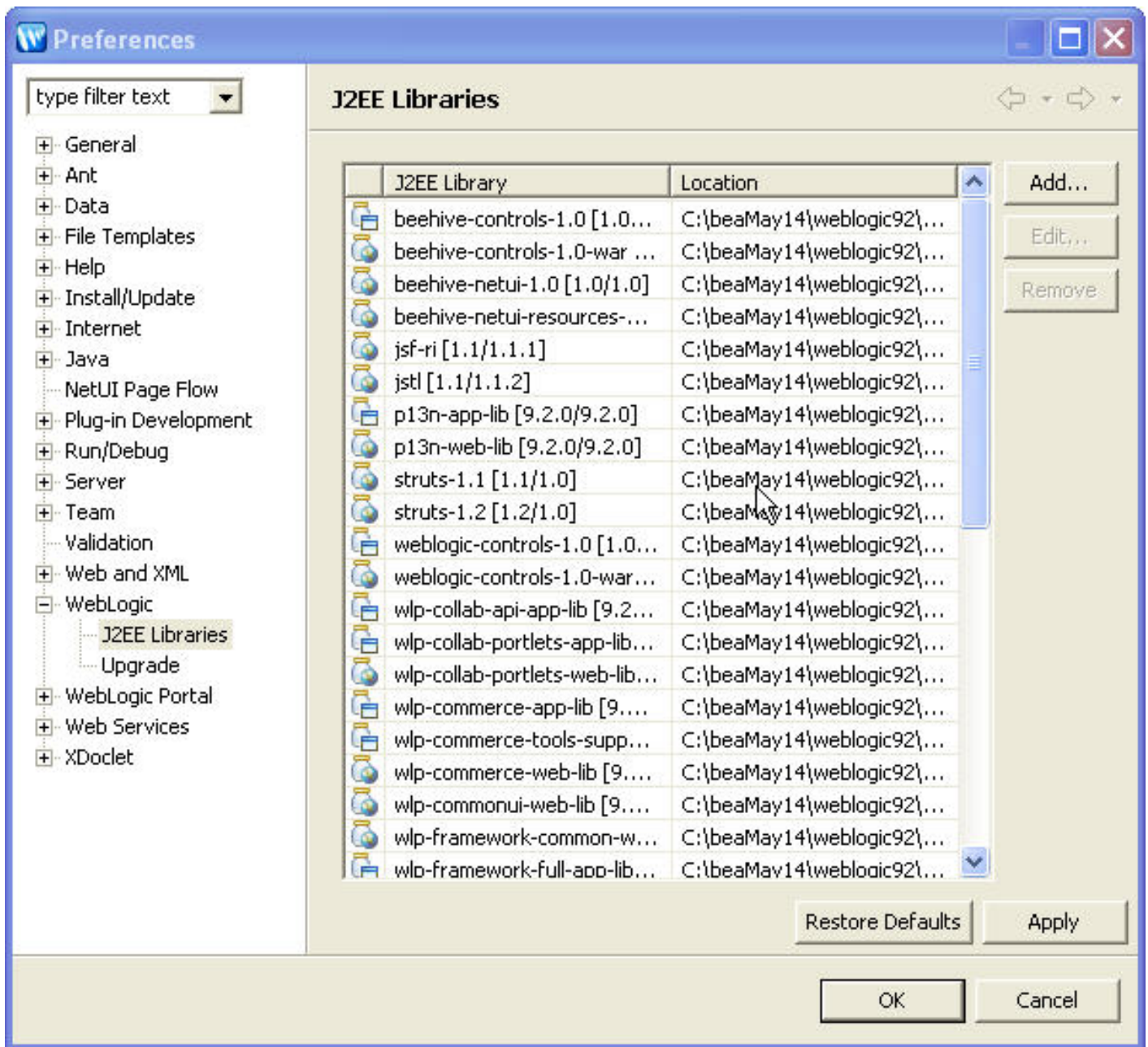


This setting is persistent and affects how WebLogic J2EE shared libraries are handled if you later add/remove facets using the **Project Facets** properties page.

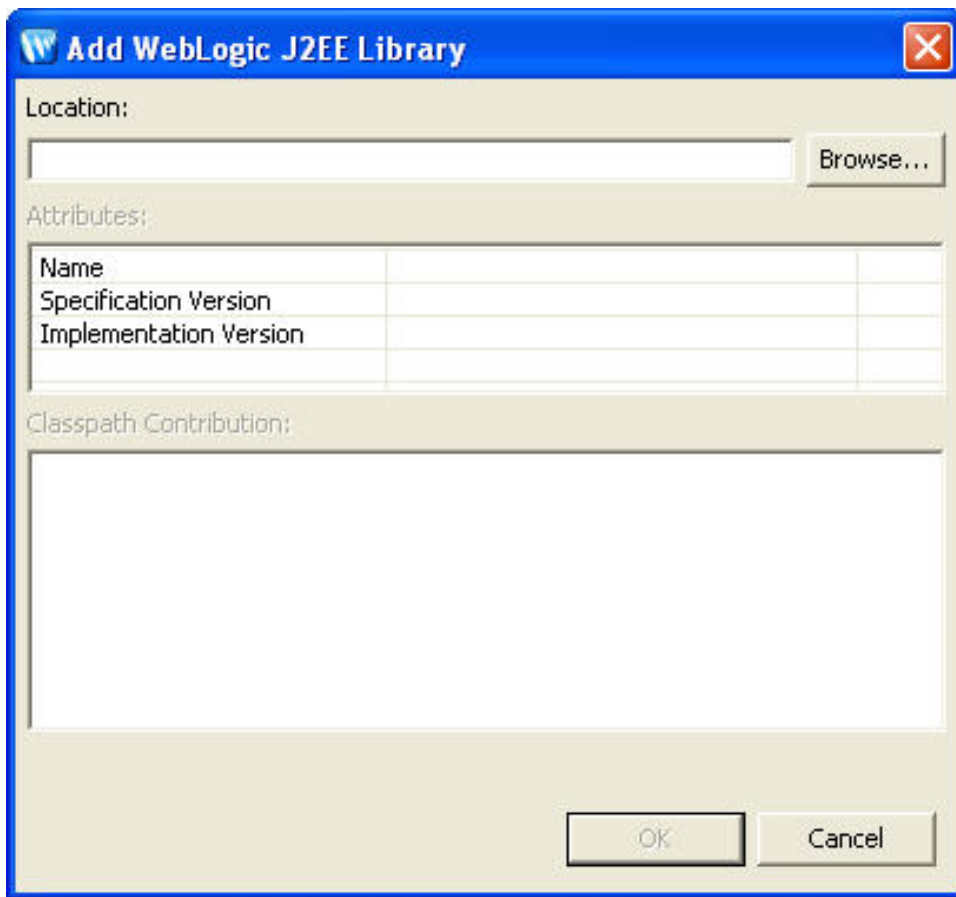
Add Custom J2EE Libraries to your Projects

You may have access to other J2EE libraries that you want to use in more than one project. You can use these J2EE libraries in your projects by doing the following:

1. Register the library with Workshop for WebLogic by clicking **Window > Preferences > Weblogic > J2EE Libraries**.

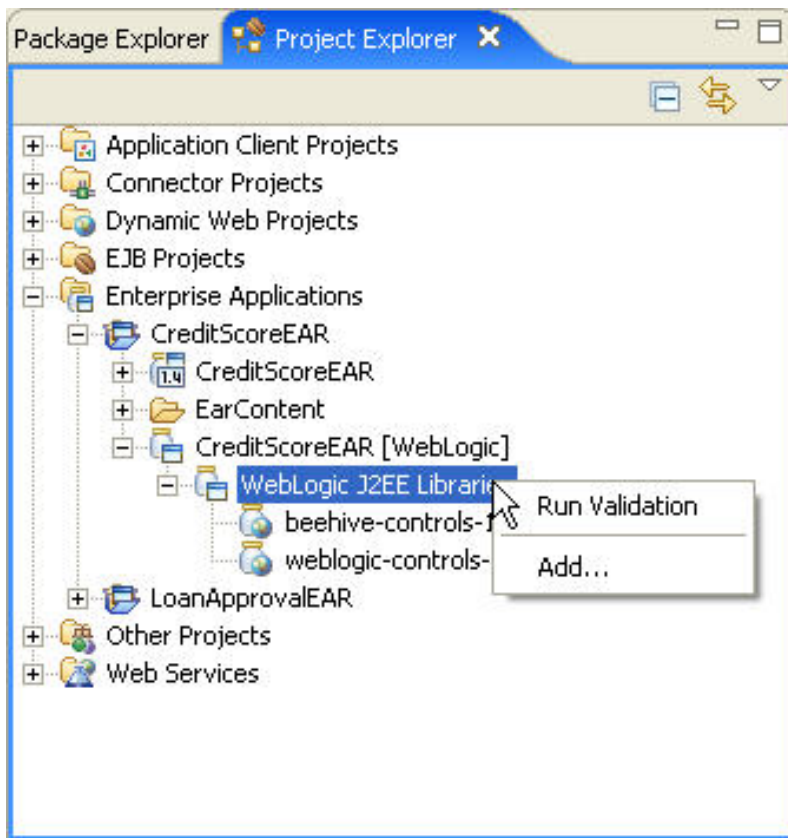


Click **Add** to add a new library:



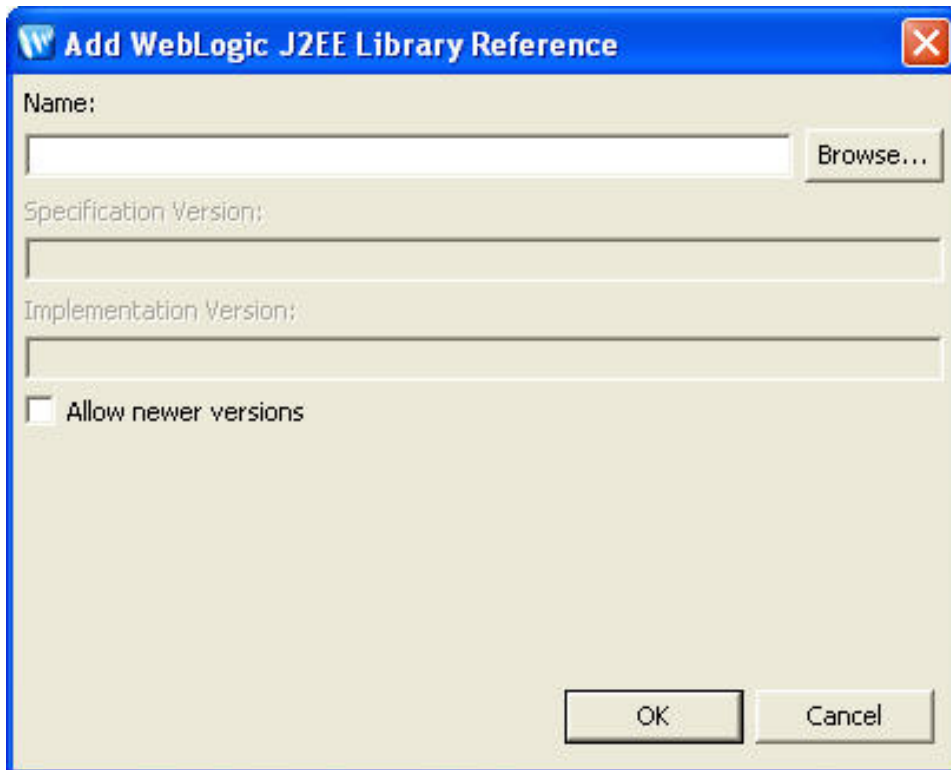
When you have entered the library location, click **OK** to finish.

2. Open **Project Explorer** view by clicking **Window > Show View > Other > Basic > Project Explorer**.
3. For applications, expand **Enterprise Applications** in the **Project Explorer**. For projects, expand the project type (i.e., **Dynamic Web Projects** for web applications, **Web Services** for web service projects, etc.)
4. Expand the project name.
5. Expand the **[WebLogic]** object. For example, if your project is called **MyEAR**, you would expand **MyEAR [WebLogic]**
6. Right click on the **WebLogic J2EE Libraries** object as shown below.



Click **Add** from the popup menu.

7. Enter the library path/name or browse for the library name. Click **OK** to accept.



Related Topics

none.

Customizing Project Dependencies

This topic describes advanced techniques for managing the location and availability of shared objects in your enterprise applications.

Web projects (web service or dynamic web projects) may have dependencies on either of the following:

- EJB projects (which are built into J2EE modules for deployment)
- Utility projects (which are built into JAR files for deployment)

These project linkages can be created in various ways. You can:

- [Make Utility Projects Available to Web Projects](#)
- [Make EJB Projects Available to Selected Web Projects in an EAR](#)
- [Adding Web Projects to Multiple EAR Projects](#)

Making Utility Projects Available to Web Projects

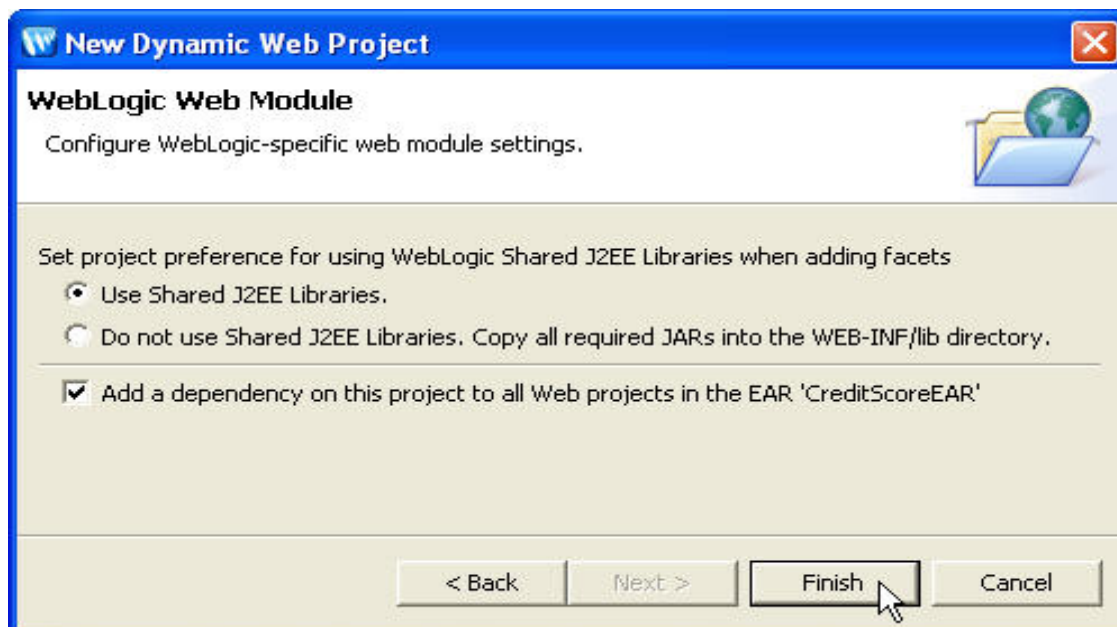
All utility projects in an EAR project are automatically compiled into JARs that are copied into the APP-INF/lib directory of the deployed EAR.

As an alternative, you may specify that the utility project should be compiled into the WEB-INF/lib folder of a web project where it will be available only to that project.

A utility project can be added to more than one EAR project and available to more than one web project.

Option 1: Making Utility Projects Available to Web Projects at Project Creation

When creating a web services project or dynamic web project, if you choose **Add project to an EAR** on the first screen of the wizard, then one of the screens of the project creation wizard has the following default settings:



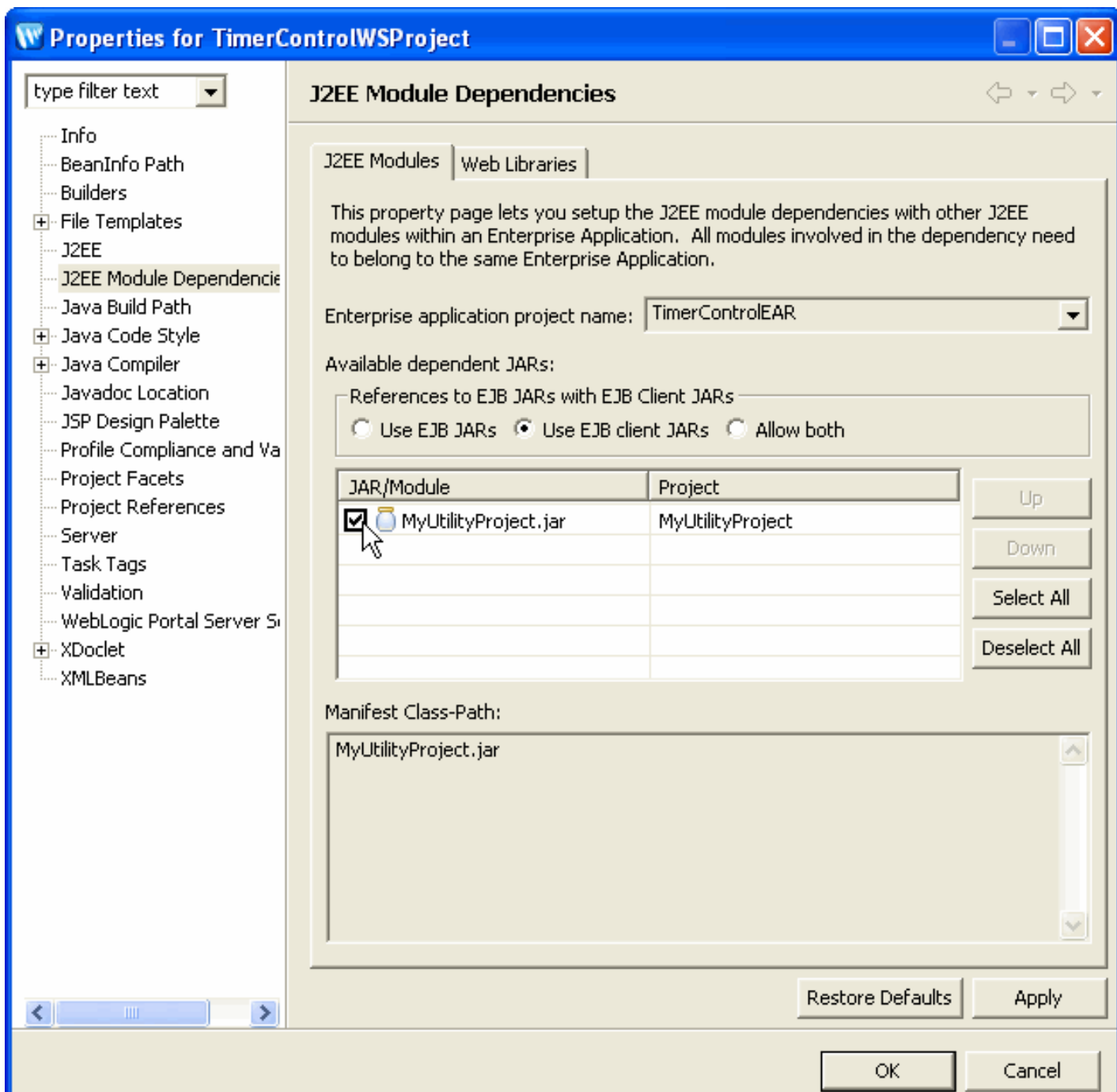
Note that if you did **not** choose **Add project to an EAR** on the first screen of the creation wizard, then the **Add a dependency from this project to all Web projects in the EAR** option will not be displayed and you must add the project to an EAR manually.

The default setting **Add a dependency from this project to all Web projects in the EAR** specifies that ALL of the utility and EJB projects will be available to this project when it is built. To make selective dependencies to specific projects, uncheck the **Add a dependency from this project to all Web projects in the EAR** option. You must then make any needed utility or EJB projects available explicitly.

Option 2: Adding a Utility Project to a Specific Web Project

When a utility project is already added to the EAR, you can add that utility project to a web project:

1. Right click on the web services or dynamic web project name in the **Package Explorer** view and choose **Properties** from the popup menu. Click on **J2EE Module Dependencies** in the bar at the left. Be sure that the **J2EE Modules** tab is selected.



OK

Cancel

2. Check the box beside the name of the utility project. Click **Apply** to save the change.

After the utility project is added, it will appear on the build path for this web project and the utility project types will be available in the IDE. Note that the **Java Build Path** option on the **Properties** dialog does not set up project dependencies correctly for Workshop for WebLogic projects. Adding a utility project to a web project means that the web project will access the shared copy of the utility module in the EAR. No files will be copied to the web project.

Note that the **References to EJB JARs with EJB Client JARs** box and the **Use EJB JARs**, **Use EJB client JARs** and **Allow both** buttons are not used by Workshop for WebLogic. Note also that EJB projects do not appear on the **Web Libraries** tab because EJBs are not libraries.

Option 3: Moving a Utility Project from an EAR Project to a Web Project

To add a utility project to a web project:

1. Right click on the web services or dynamic web project name in the **Package Explorer** view and choose **Properties** from the popup menu. Click on **J2EE Module Dependencies** in the bar at the left. Be sure that the **J2EE Modules** tab is selected.

Properties for TimerControlWSPProject

type filter text

- Info
- BeanInfo Path
- Builders
- File Templates
- J2EE
- J2EE Module Dependencies**
- Java Build Path
- Java Code Style
- Java Compiler
- Javadoc Location
- JSP Design Palette
- Profile Compliance and Va
- Project Facets
- Project References
- Server
- Task Tags
- Validation
- WebLogic Portal Server S
- XDoclet
- XMLBeans

J2EE Module Dependencies

J2EE Modules | Web Libraries

This property page lets you setup the J2EE module dependencies with other J2EE modules within an Enterprise Application. All modules involved in the dependency need to belong to the same Enterprise Application.

Enterprise application project name: TimerControlEAR

Available dependent JARs:

References to EJB JARs with EJB Client JARs

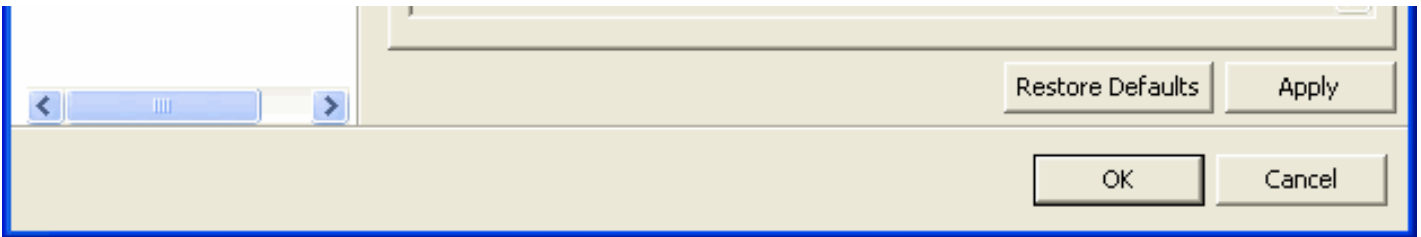
Use EJB JARs Use EJB client JARs Allow both

JAR/Module	Project
<input checked="" type="checkbox"/> MyUtilityProject.jar	MyUtilityProject

Up
Down
Select All
Deselect All

Manifest Class-Path:

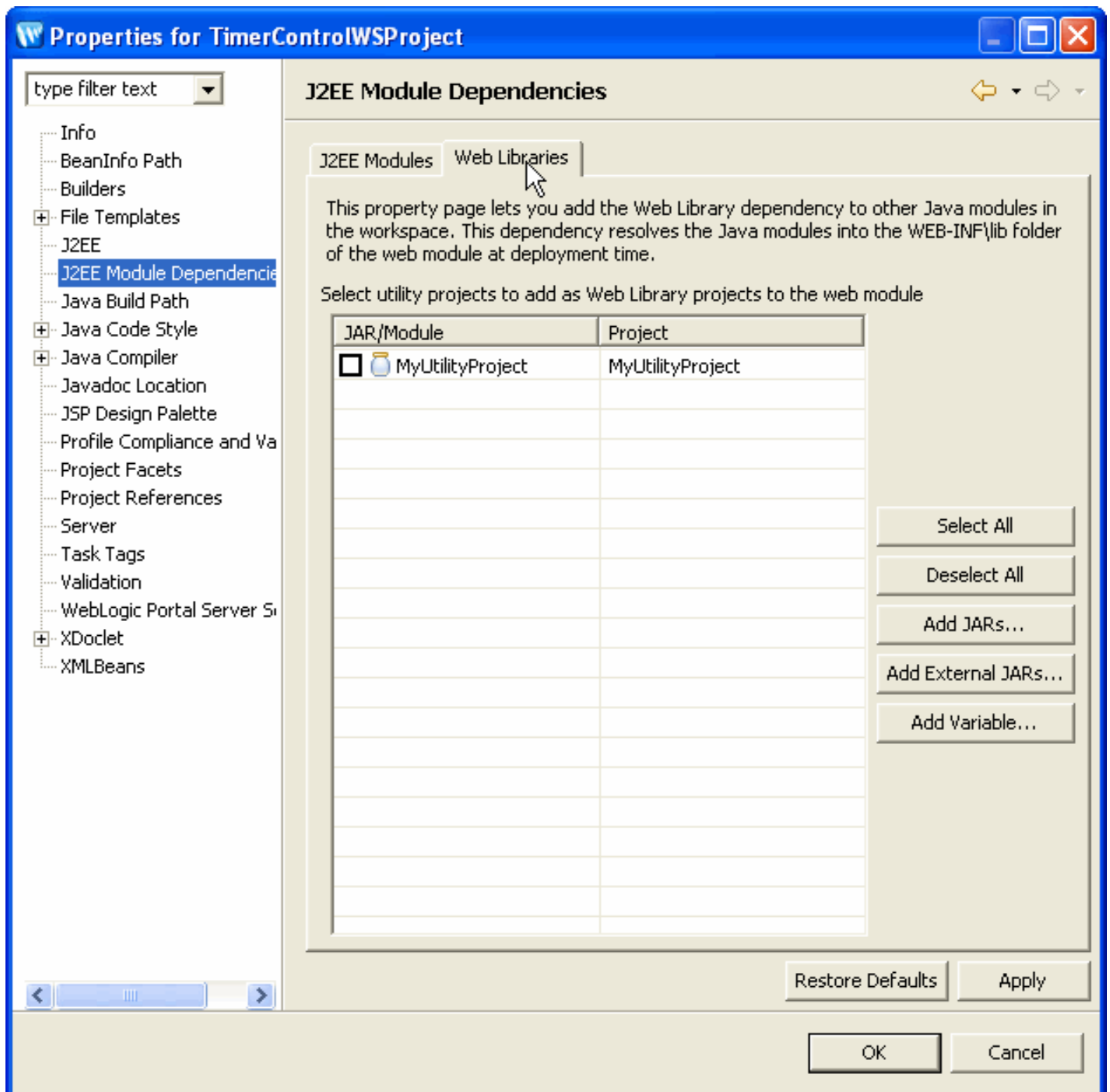
MyUtilityProject.jar



Uncheck the box beside the name of the utility project. Click **Apply** to save the change.

Note that the **References to EJB JARs with EJB Client JARs** box and the **Use EJB JARs, Use EJB client JARs** and **Allow both** buttons are not used by Workshop for WebLogic. Note also that EJB projects do not appear on the **Web Libraries** tab because EJBs are not libraries.

- Now click the **Web Libraries** tab. In the sample below, there is a new utility project which is not added to the current project.

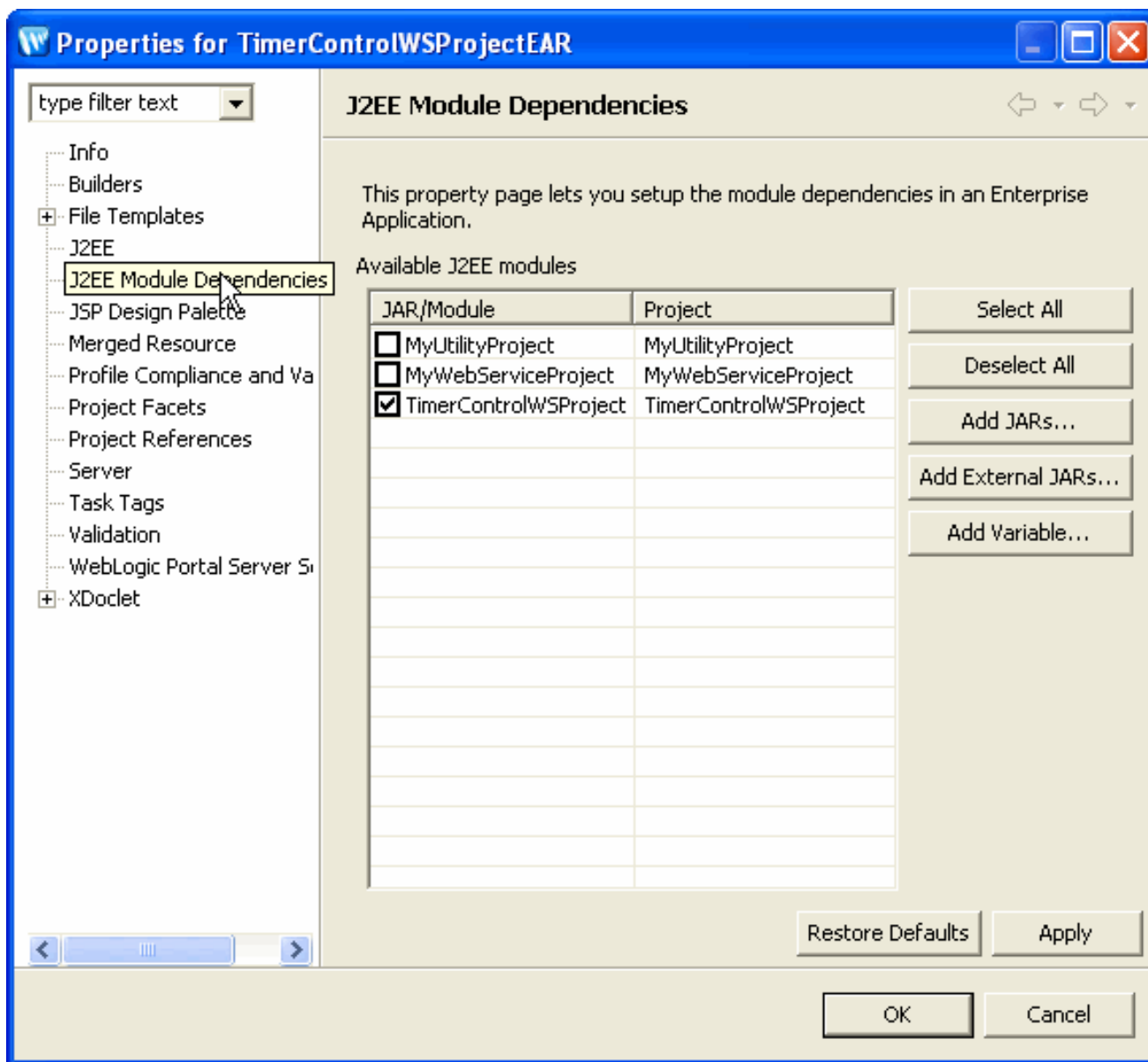


- Click in the box beside the project name and click **OK** to add the new utility project to the existing project and make a virtual copy of its JAR file in the WEB-INF/lib folder of this project so that it will be available for build/deployment.

The J2EE Module Dependencies option is the recommended option for creating project dependencies in Workshop for WebLogic. Do not use the Java Build Path option to set project dependencies because it does not create all dependencies correctly.

Now you must stop the utility project from compiling into the EAR's APP-INF/lib directory by:

- Select the EAR project by clicking on the EAR project name in the **Package Explorer** view.
- Click **Project > Properties** and from the **Properties** dialog, click on **J2EE Module Dependencies** in the bar at the left.

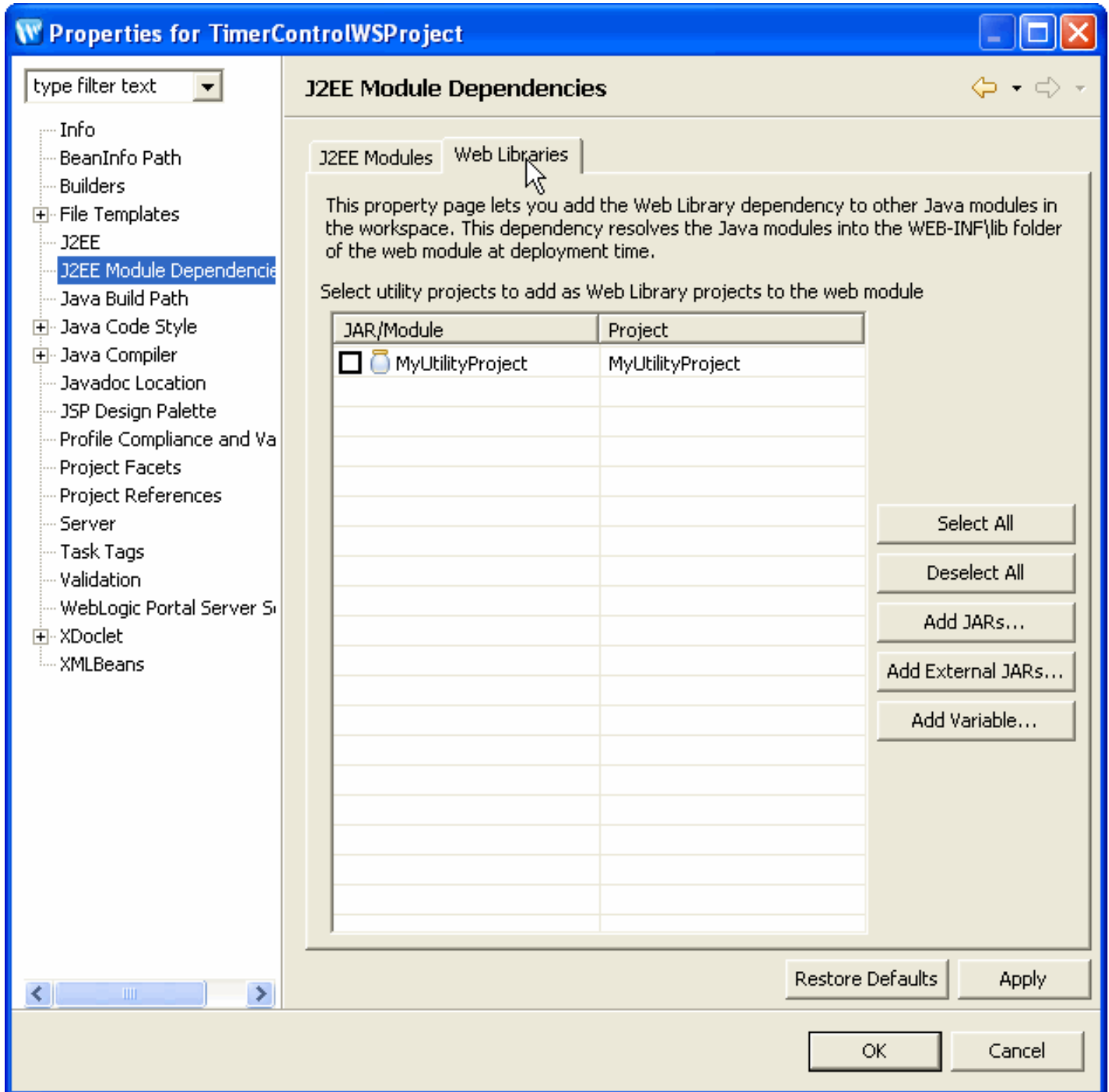


- Click in the box beside the project name to REMOVE the check mark and click **OK** to remove the utility project from the EAR project.

Option 4: Adding A Utility Project from Outside the EAR

A utility project can also be added to a web project in an EAR even though the utility project is not added to the EAR itself. To do that:

1. Right click on the web services or dynamic web project name in the **Package Explorer** view and choose **Properties** from the popup menu. Click on **J2EE Module Dependencies** in the bar at the left. Click the **Web Libraries** tab. In the sample below, there is a new utility project which is not added to the current project.



2. Click in the box beside the project name and click **OK** to add the new utility project to the existing project and copy its JAR file to the WEB-INF/lib folder of this project on deployment.

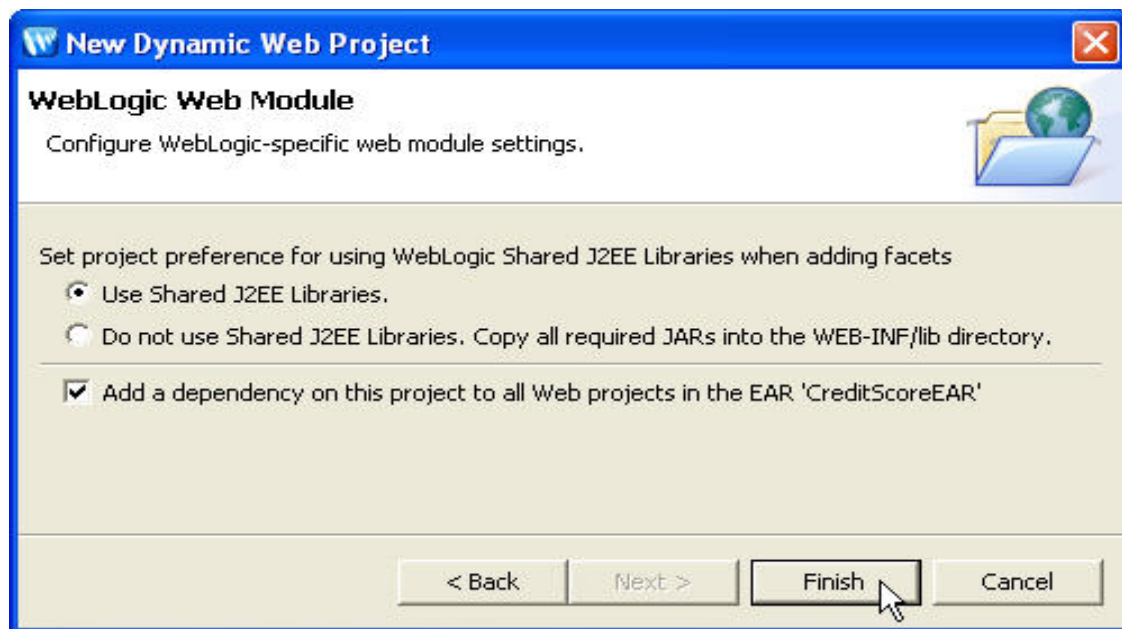
Utility projects that are not added to the EAR can ONLY be added through the WEB-INF/lib folder of individual web projects.

Making EJBs Available to Selected Projects in an EAR

All EJB projects in an EAR project can be available to web services or dynamic web projects.

Option 1: Adding EJBs to Web Projects/Modules at Project Creation

When creating a web services project or dynamic web project, if you choose **Add project to an EAR** on the first screen of the wizard, then the fourth screen of the wizard has the following default settings:



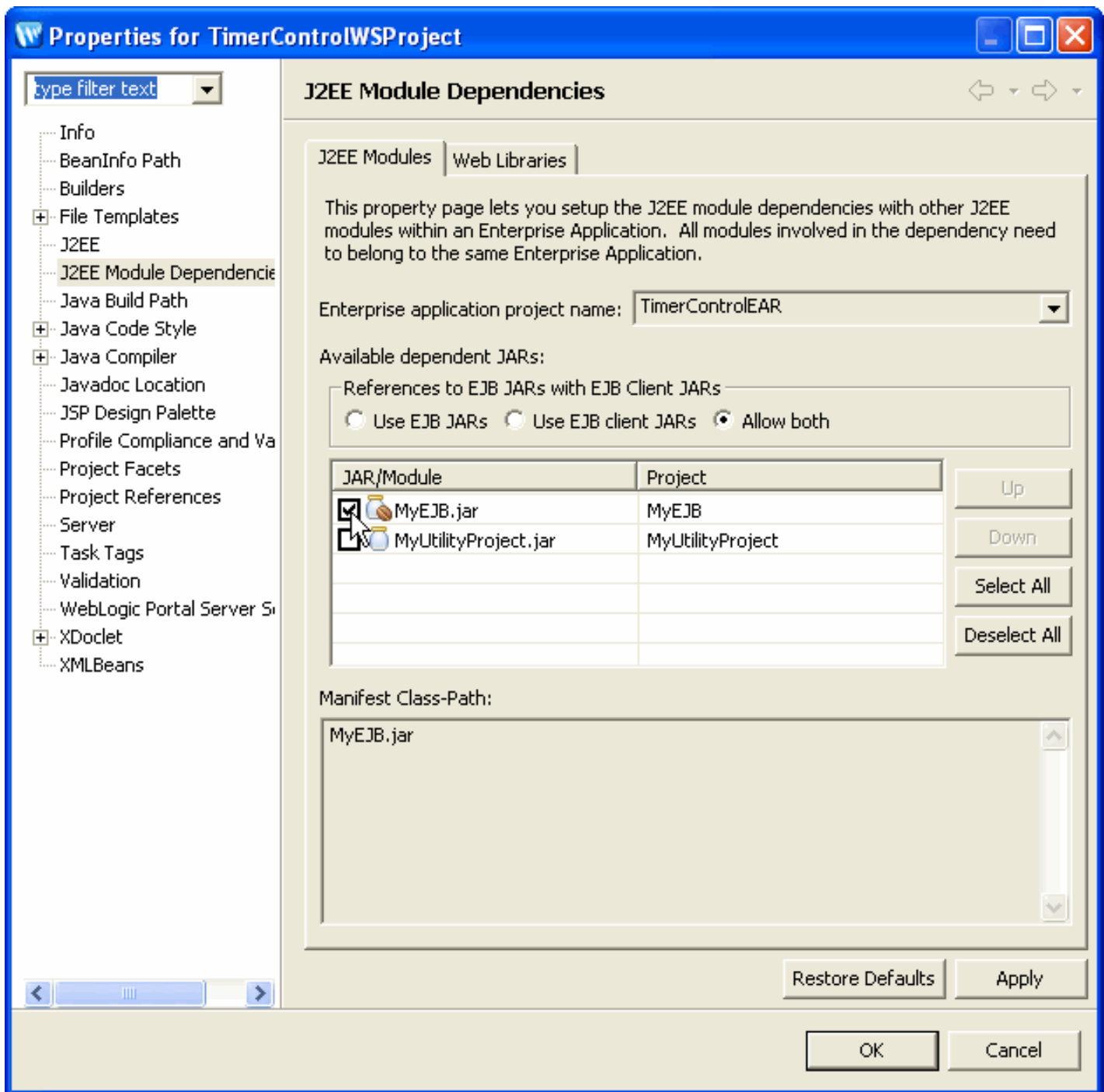
Note that if you did **not** choose **Add project to an EAR** on the first screen of the creation wizard, then the **Add a dependency on this project to all Web projects in the EAR** option will not be displayed and you must add the project to an EAR manually.

To make selective dependencies to specific EJB projects, uncheck the **Add a dependency on this project to all Web projects in the EAR** option. You must then add any needed projects explicitly.

Option 2: Adding an EJB to a Web Project

If an EJB project has been added to an EAR, you can add the EJB project to web projects in the EAR by doing the following:

1. Right click on the web services or dynamic web project name in the **Package Explorer** view and choose **Properties** from the popup menu. Click on **J2EE Module Dependencies** in the bar at the left. Be sure that the **J2EE Modules** tab is selected.



Check the box beside the name of the EJB project. Click **Apply** to save the change.

Note that the **References to EJB JARs with EJB Client JARs** box and the **Use EJB JARs**, **Use EJB client JARs** and **Allow both** buttons are not used by Workshop for WebLogic. Note also that EJB projects do not appear on the **Web Libraries** tab because EJBs are not libraries.

Option 3: Making an EJB from Outside the EAR Available

If you are using a standalone EJB that is available at runtime on the WebLogic Server but not deployed with your EAR project, you must set it using the **Java Build Path** property page.

1. Right click on the project name in the **Package Explorer** view and choose **Properties** from the popup menu. Click on **Java Build Path** in the bar at the left.

Note that the **Java Build Path** property page affects only Java source editing within the IDE. It does not affect deployment, exporting J2EE archive files, or generating Ant scripts.

Adding Web Projects to Multiple EAR Projects

If you have a web project that you use in more than one EAR, you can simply open each EAR project and [link the project to that EAR](#).

When a project is a member of more than one EAR project, the **Enterprise application project name** pulldown box on the **J2EE Module Dependencies** property page will list the names of the EAR projects.

If a project is added to multiple EAR projects and you deploy more than one EAR project to the **same server**, there will be a name collision because there will be two copies of the same project deployed to the server. To eliminate the collision, you may [undeploy one of the projects](#).

Alternately, you may open one of the EAR projects and edit the file `EarContent/META-INF/application.xml` . Change the value of the `<context-root>` element for the web project to a unique value. Changing this file is an advanced technique and should be done with care.

Related Topics

[Configuring Standard Project Dependencies](#)

Understanding the Build Process

Before building your application, your application components must be assembled into projects. Projects and their usage are described in [Applications and Projects](#). Review the information in [Managing Project Dependencies](#) to understand project dependencies and making JARs and J2EE libraries available to your projects.

Under normal circumstances, you will never explicitly build your projects/applications since by default, Workshop for WebLogic builds your files automatically, using the Eclipse **Build Automatically** feature. This feature does an incremental build whenever a program file is saved.

If you wish, you can build your files, projects and enterprise applications manually. Workshop for WebLogic uses the standard Eclipse commands for building, as described in detail in the Eclipse/JDT documentation, available by clicking **Help > Contents** and choosing **Java Development User Guide** from the **Contents** pane at the left.

Cleaning and Rebuilding

The clean operation (the **Project > Clean** command) cleans all build artifacts resulting from a previous build of a project. It's a good way to "start over" so that you know that your build output includes only the most up-to-date files based on your source code. You should clean your project before final production deployment.

Building a Production-Quality Application

Before you deploy your application to a production environment, you will probably want to perform a clean operation followed by a complete build in a controlled environment. In this way you can ensure that you have a formal, repeatable build process that you use every time you deploy an application to production.

The Build Commands

The **Project** menu provides the following standard Eclipse commands for building your files:

Command	Description
Build All	Build all open projects, including EAR projects, web projects, utility projects, EJB projects and any other open projects. This command is only available when the Build Automatically command is disabled.
Build Project	Build a single project. This command is only available when the Build Automatically command is disabled.
Build Working Set	Builds the current working set. A working set is a customized list of resources that you define using the pull-down menu from the Navigator view. Working sets are described in detail in the Eclipse documentation which you can view by clicking Help > Help Contents > Workbench User Guide . This command is only available when the Build Automatically command is disabled.
Clean	Removes build artifacts and optionally rebuilds selected project(s).
Build Automatically	This is the default setting. Clicking on this command toggles automatic build mode.

Related Topics

[Creating Custom Ant Build Files for an Application](#)

[Deploying, Running, and Debugging Applications](#)

Setting up Servers for Use Within the IDE

This topic explains how add a WebLogic server domain to the Workshop for WebLogic development environment, so that you can easily deploy, run, and test projects on the server.

This topic also explains how to add and remove individual projects from a server domain.

The topic is divided into two sections:

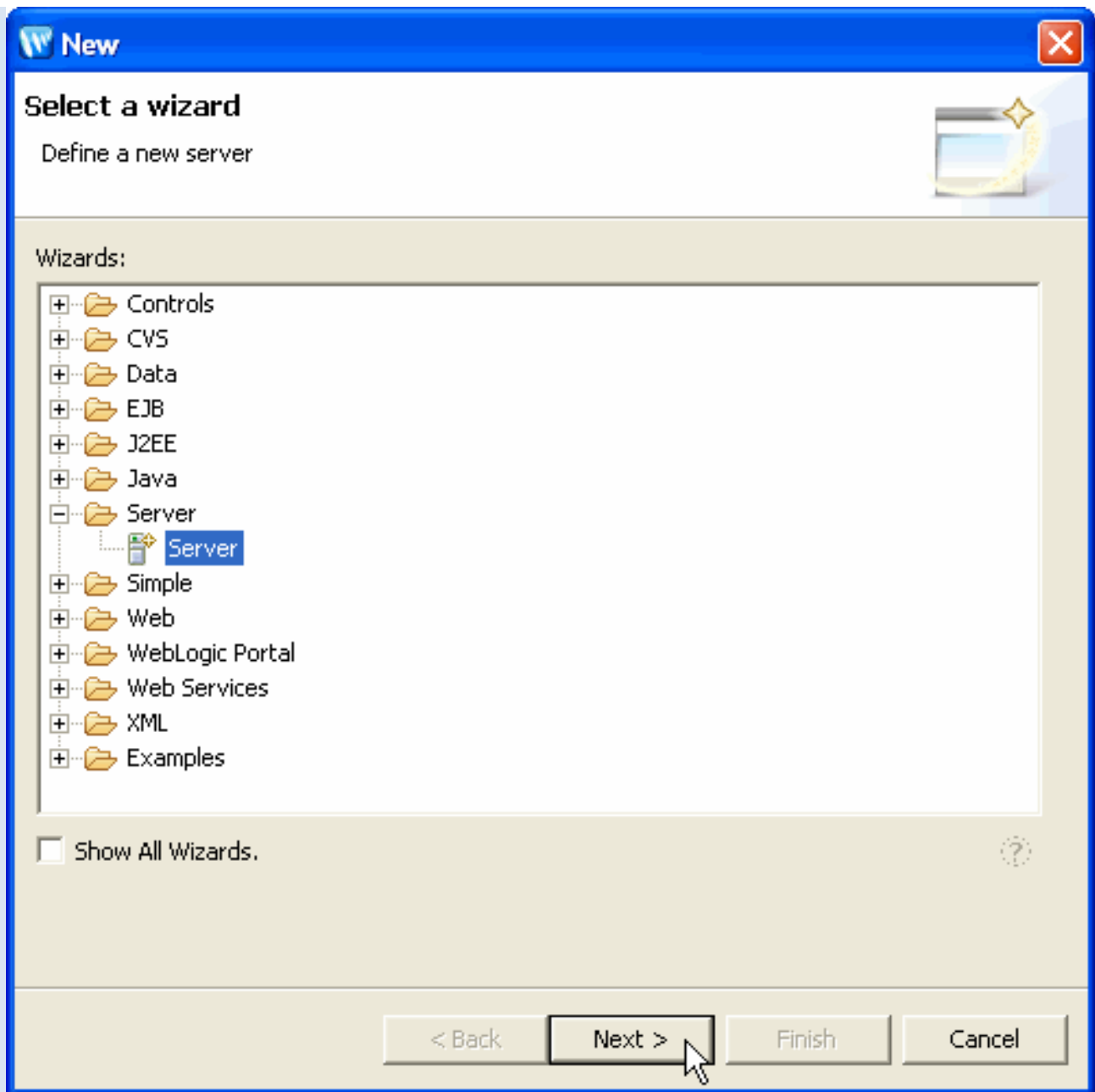
- [Defining a Server for Use With the IDE](#)
- [Adding and Removing Projects](#)

Defining a Server for use within the IDE

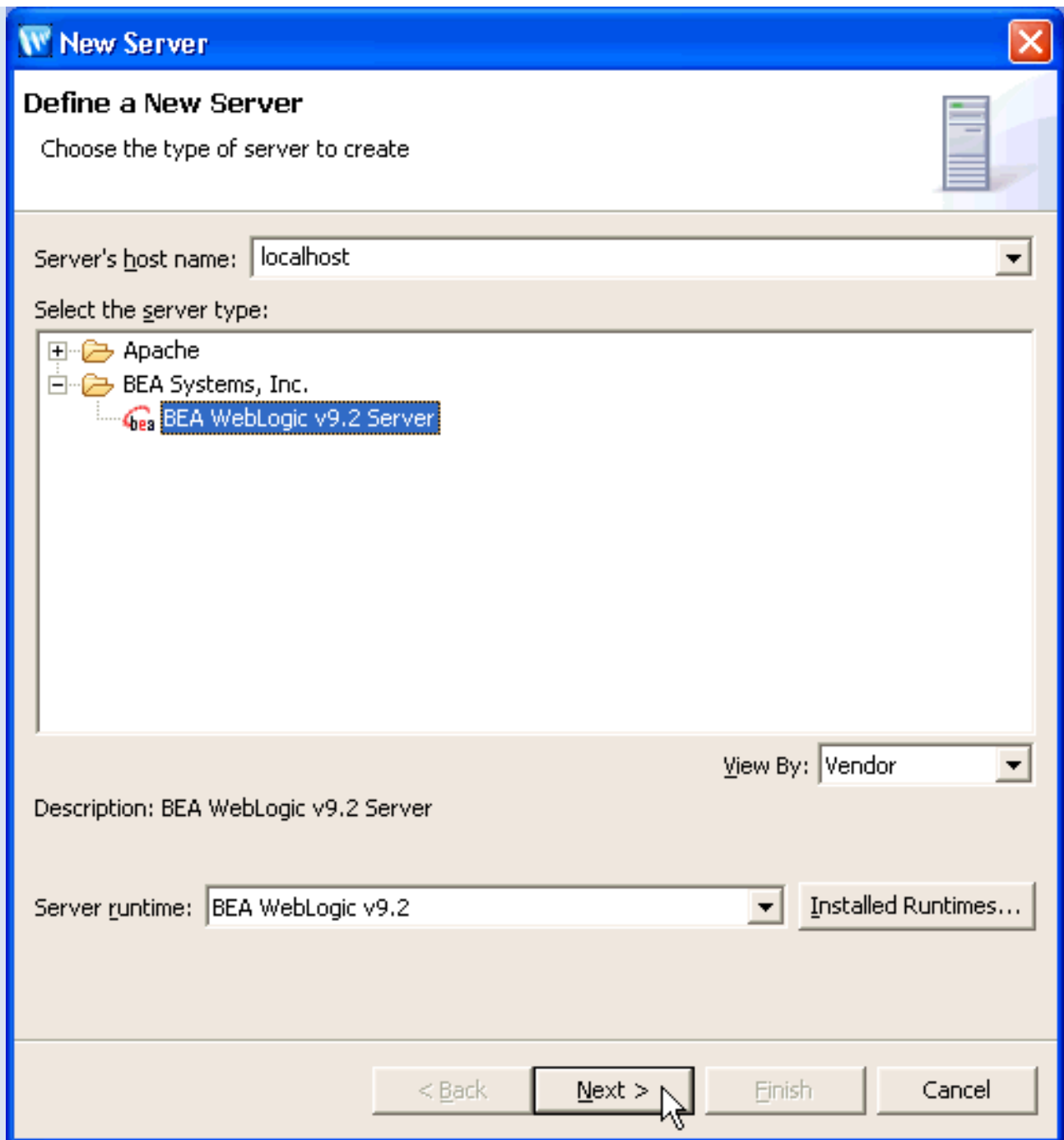
From the IDE perspective, a *server* is an link/pointer to a local WebLogic Server domain. You must define a server within the IDE in order to run and test your applications. If you try to run your application without first defining a server, you will be prompted to create a server in a series of dialogs similar to the steps below.

To define a server:

1. Click **File > New > Other**. (Alternatively, right-click anywhere within the **Servers** view and select **New > Server**.)
2. Expand **Server**, click **Server** from the expansion list, then click **Next**.



3. Be sure that the **BEA Systems, Inc** node is expanded and **BEA WebLogic v9.2 Server** is highlighted.
Click **Next**.



- Click the drop-down field **Domain home** and select the default Workshop for WebLogic server domain at the following location:

<BEA_HOME>\weblogic92\samples\domains\workshop

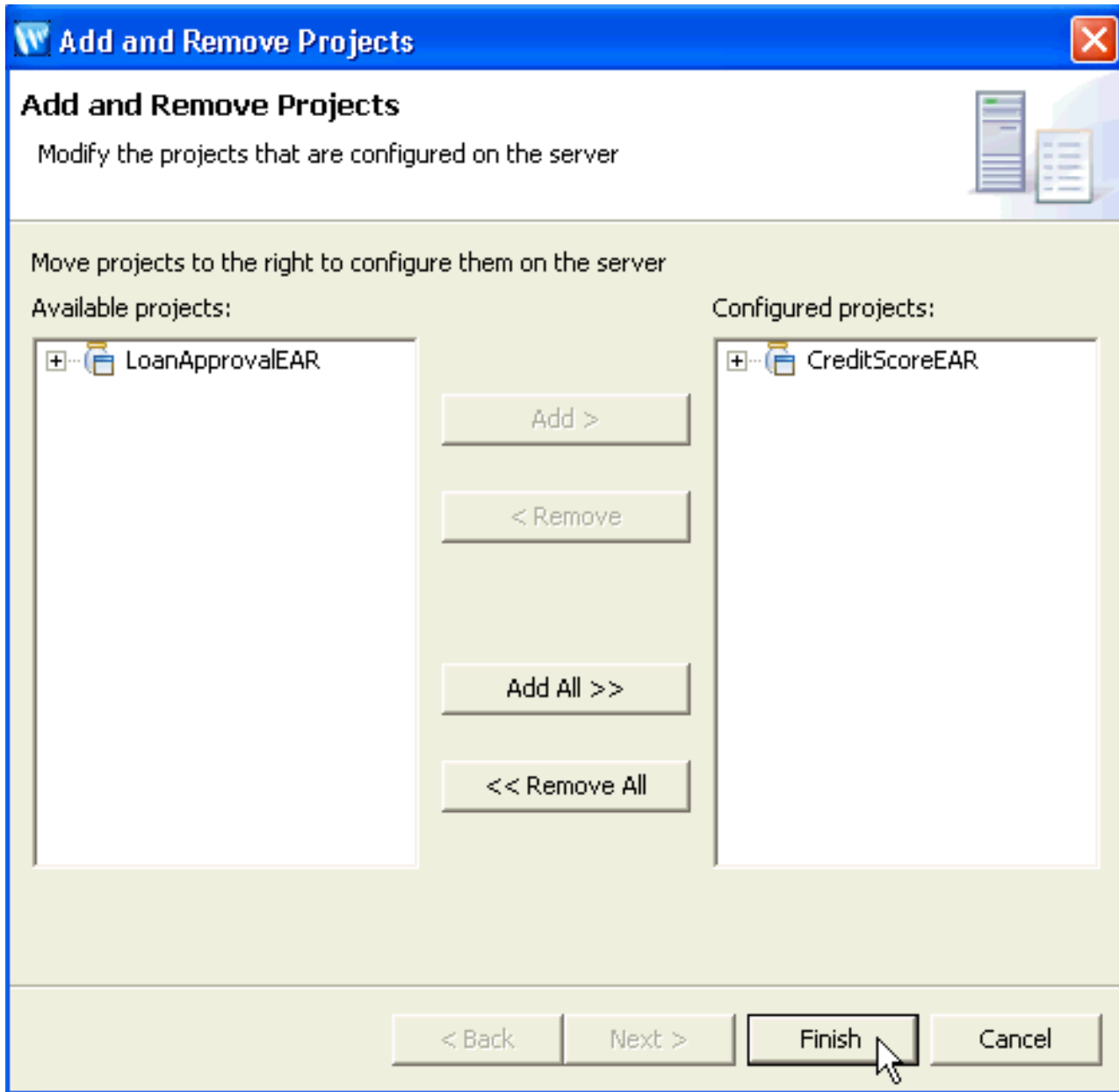
This domain already contains all of the resources necessary to deploy and run Workshop for WebLogic applications. (Alternatively, you can use the **Browse** button to select a different server domain.)

Click **Finish**.

The new server definition will be added to the **Server** view.

Adding and Removing Projects from the Server

To add or remove projects from a server, on the **Server** view, right-click the target server, and select **Add and Remove Projects**.



The **Add and Remove Projects** dialog shows a list of project available for addition to the server. Select and move projects from the left column to add them to the server.

Servers will retain any deployed projects that were previously run on them, even once they have been turned off. Stopping and restarting the server does not remove previously deployed projects.

Projects in Workshop for WebLogic become J2EE modules on the server. If your current application includes projects with the same name as previously deployed projects, the previous application's project modules will remain on the server and potentially create conflicts with new projects/modules with the same name.

To remove previously added projects, move the project from the right column to the left column.

Related Topics

When working with large applications, you may want to carefully manage file deployment to the server to improve performance. More advanced server management is described in [Managing Servers](#).

[Deploying, Running, and Debugging Applications](#)

[Creating a Workshop for WebLogic-Enabled Server Domain](#)

Managing Servers

When you install Workshop for WebLogic, a copy of WebLogic Server is installed on your computer. You can then do iterative development by running your application(s) on this local server. However the server and the files that you deploy to the server must be managed to prevent server conflicts and unexpected behavior.

Overview of Server Management Tasks

Do Once for Each Workspace:

Create a server definition in the IDE. A *server* is simply a link/pointer to your server (the domain on the local WebLogic Server). You may need multiple servers if you need to deploy applications with different server settings. But all servers that point to a domain are, in effect, the same server.

If you define more than one server on a single domain, you must not run the servers simultaneously because servers typically use the same port number and requests to different servers on the same port will result in unpredictable behavior.

Do Each Time you Switch Workspaces

Clear duplicate project/modules from the server. The Run/Debug commands make a pointer to your projects on the server. This pointer persists on the server, and must be managed for best performance. When you begin to work on a different application, old projects (which became J2EE modules on deployment to the server) with duplicate names must be removed.

Do to Speed Up Testing

Manage the projects that are deployed to the server. To enhance testing performance, you can choose to limit the projects that are deployed to the server.

Creating a Server Definition in the IDE

To create a new server definition to use within the IDE, see the topic [Setting up Servers for Use Within the IDE](#).

Each server definition within a domain uses the same default port number and address. New server definitions will retain any deployed projects. Deleting a server definition and creating a new one does not clear deployed projects from the remaining server definitions.

Note that defining a second servers in a single domain creates by default, another server with the **same port number**. If you have two servers defined, you should not run your applications on both at the same time, since the servers will experience conflicts and produce unpredictable results.

Note also that projects that have been deployed in the past may remain on the server until undeployed. Running new applications on the server may result in conflicts if the new applications include projects with the same names as projects in previous applications. To prevent conflicts, you may want to create different domains for servers that will be used for testing different enterprise applications.

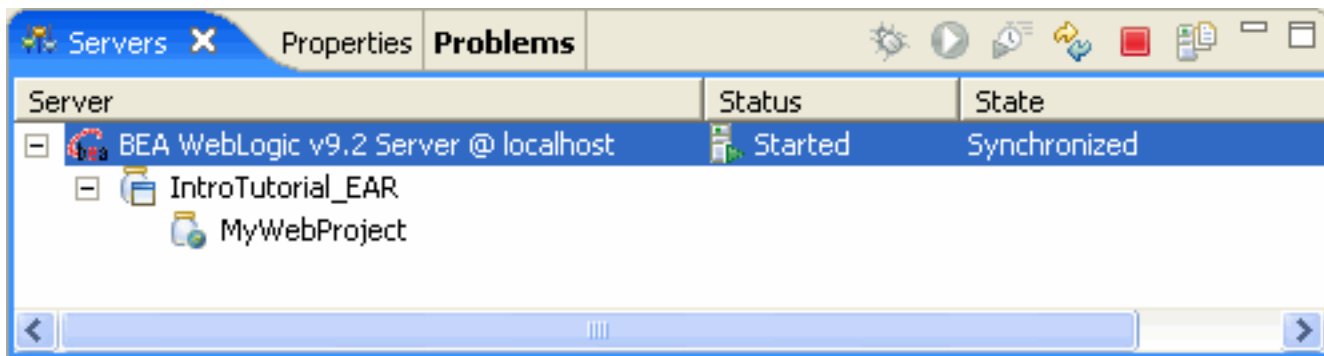
Undeploying Previously Deployed Projects

To undeploy previously deployed projects from a server see the topic [Adding and Removing Projects from the Server](#).


Stopping and restarting the server does not remove application files. Opening a new workspace and defining a new server does not remove application files.

Managing the Projects that are Deployed

From the **Servers** view, expand the name of your server. The projects from the current workspace that are associated with this server will be displayed. This is not an indication that the projects are actually deployed to the server, simply that they are associated with the server.



If you right click on the server, you can click **Add and Remove Projects** to dissociate a project from the server and prevent it from being automatically deployed when the application runs.

Note that only **open** projects are deployed. You can close a project by clicking on the project folder name in the **Navigator** or **Package Explorer** view and then click **Project > Close Project**. The project folder icon will change to  to indicate that the project is now closed. Closed projects are not deployed. Note that if the project was deployed *before* it was closed, it will remain on the server unless you undeploy it.

You can also use the *working sets* feature of Eclipse to specify a collection of projects/applications to be deployed as a set. You can get information on working sets from the help system by clicking **Help > Help Contents** and choosing **Workbench User Guide**.

If you delete projects from your current application, those projects will remain on the server and must also be removed manually.

Related Topics

From <http://edocs.bea.com> click on the link for documentation for WebLogic Server to get complete documentation for the **Configuration Wizard** and domain creation. This documentation set also provides a complete discussion of when you might want to have more than one domain.

[Setting up a Server](#) describes how to define a server and add and remove projects from the server.

[Deploying, Running, and Debugging Applications](#)

Deploying, Running, and Debugging Applications

Under normal circumstances, Workshop for WebLogic builds your files automatically and you can deploy at any time. **Before you can deploy your application, you must have a successful build completed.**

The following topic explains the basic concepts of deploying and debugging Workshop for WebLogic applications to WebLogic Server running in development mode.

Development Mode

Weblogic Server can be started in one of two modes: development or production mode. (When you start the Workshop for WebLogic server domain using the startup script provided, it runs in development mode.) In development mode, WebLogic Server behaves in ways that make it easier to iteratively develop and test an application: for example, it automatically deploys the current application in an exploded format and certain security restrictions on deployment are relaxed.

What Happens During Deployment

Deployment consists of three steps:

1. Assembling (e.g., EJB generation and JWS compilation) if required.
2. Publishing the files to the server.
3. Running the application.

These three steps are normally done seamlessly. An EAR file is not generated for deployment to a single test server. The table below shows how different applications and application components are deployed both for development and for production.

Development Aim:	Implementation:	Deployment by Workshop for WebLogic:	
		for Iterative Test/Debug (Local WebLogic Server)	for Production Deployment (WebLogic Production Server)
Enterprise Application	EAR project	exploded EAR (split source)	.EAR file
Web Artifacts (JSPs, web services, web applications)	Dynamic web project or web service project	exploded .WAR (split source)	.WAR file
Shared code	Utility project	virtual JAR mapped into WEB-INF/lib or APP-INF/lib	JAR copied into WEB-INF/lib or APP-INF/lib
Enterprise JavaBean	EJB project	J2EE EJB module	J2EE EJB module
External objects available to projects	build path	class path	class path
J2EE libraries (also called shared J2EE libraries)	JARs available on build path	virtual merge of libraries with web modules that use them	virtual merge of libraries with web modules that use them

What Gets Deployed

When you deploy, *all* open projects associated with the server are deployed. You can use the **Project > Close Project** command to close projects. You can also specify working sets (described in the Eclipse help system in the **Workbench User Guide**) to control how much gets built and deployed.

However, even though an entire application or group of projects was deployed, only the file/folder that you clicked on (to initiate the deploy) will display its results. For example, if you deploy from a page flow component file, that page flow will run in a new browser tab in the editor area. If you deploy a web service, the test client page for that web service will run in a new tab in the editor area. If the page flow or web service relies on other web services to run correctly, it will still work because all components are actually deployed.

You can also manage the projects that get deployed to the server.

Deploying your Application

There are two commands that deploy your application:

1. Click **Run As > Run on Server** to deploy your application. This command starts the server in normal mode (if it is not already started), does pre-assembly, and runs the application.

- Click **Debug As > Debug on Server** to deploy your application in debug mode. This command starts the server in debug mode (if it is not already started), does pre-assembly, and runs the application.

If you wish to simply publish your files to the server, you can use the **Servers** view (described [below](#)) to publish only. The **Servers** view also allows you to un-deploy applications from the server.

Viewing the Results of Deployment

After you deploy your application, the **Servers** view is displayed automatically and the results of running your application are displayed in the editor area of the workbench.

If you are running a page flow, the initial JSP page will appear in a tab in the editor area. This tab is a test browser that allows you to run your application.

If you are running a web service, the test client appears in a tab in the editor area. This tab allows you to specify the parameters to an operation and make a request to that operation. The response from the operation is displayed in the same tab.

Deploying Applications and Managing Servers with Servers View

The **Servers** view opens automatically when you deploy/run your application. You can also open it manually by clicking **Window > Show View > Other**, expand **Server** and click **Servers**. There are many commands available from the **Servers** view for managing your server (s), as shown in the diagram below.

1. Click an icon on the toolbar:

Publish all open projects associated with the current server (i.e., perform a deploy to the server, without the final step of running the application)

Stop the server

Restart the server

Start the server in profiling mode

Start the server in normal mode

Start the server in debug mode

2. Expand the server name to view all of the projects that are associated with the server

3. Right click on the server name to see a menu of choices:

New - define a new server

Open - open a status window for the current server as a new editor tab

Refresh Status - refresh status if status window is open

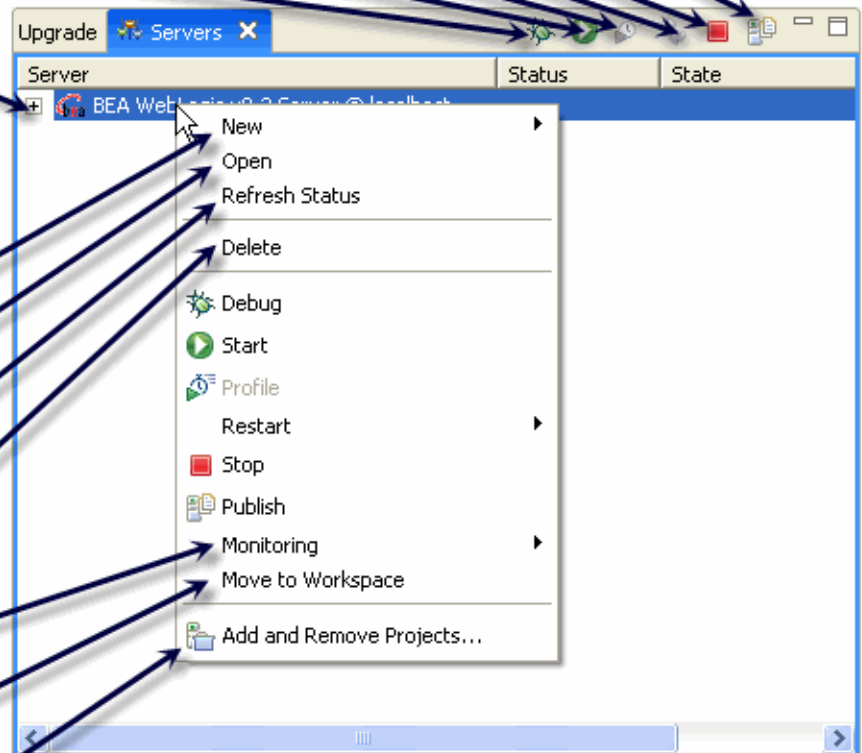
Delete - delete the server

start/stop/publish commands (same as on toolbar)

Monitoring - allows you to monitor server ports

Move to Workspace - moves the server to another workspace

Add and Remove Projects - add new projects or remove existing ones from the



Add and Remove Projects - add new projects or remove existing ones from the server

Servers view is a standard Eclipse feature, and is described in detail in the Eclipse help system which you can view by clicking **Help > Help Contents** and choosing **Web Application Development Guide** from the **Contents** pane.

Debugging your Application

The Eclipse platform provides standard debugging tools for your Java files, including setting breakpoints and displaying the values of variables. To set a breakpoint, from the editor view, right click in the marker bar (the gray bar at the left of the text) on the line where you wish to set the breakpoint and choose **Toggle Breakpoint** from the popup menu.

Once a breakpoint(s) is set, when you run the application, you will be prompted to enter **Debug** perspective and you can use the **Step Into**, **Step Over** and other commands available from the **Debug** view to step through your application. The **Breakpoints** and **Variables** views also allow you to monitor your breakpoints and variables as you debug.

The debugging commands are described in more detail in the **Java Development User Guide** section of the help system.

Deployment to WebLogic Server in Production Mode

The following procedure explains how to deploy Workshop for WebLogic applications to WebLogic Server running in production mode. This task is divided into three steps:

1. [To Ensure that Required Resources Exist on the Production Server](#)
2. [To Compile the Application into an EAR File](#)
3. [To Deploy the EAR File to the Production Server](#)

1. To Ensure that Required Resources Exist on the Production Server

Before you deploy a Workshop for WebLogic application to WebLogic Server, you must first ensure that the server domain has the resources expected by the application to be deployed. For example, Apache Beehive NetUI and Controls framework-related libraries must be present if the application uses these frameworks and the server must have the appropriate JMS queues configured if the application contains message buffering, reliable messaging, and JMS transport.

General purpose server resources for Workshop for WebLogic applications are provided through the WebLogic Server Configuration Wizard, including support for Apache Beehive NetUI and Controls, JMS, Pointbase, Java Server Pages, Java Server Faces, and Struts. See the instructions for [creating a new Workshop for WebLogic-enabled server](#) or [adding Workshop for WebLogic resources to an existing server](#).

The configuration wizard is not guaranteed to provide sufficient resources for every application. For example, if your application uses JMS queues other than the default Workshop for WebLogic queues, you will need to manually configure the server to include those non-default JMS queues. For detailed instructions on configuring JMS queues for WebLogic Server see [Configuring Basic JMS System Resources](#) in the WebLogic Server documentation. Similarly, if your application uses a datasource other than the default Pointbase datasource, you will need to manually configure the server domain to include that datasource. For more information on configuring datasources see [Configuring a Database Service](#) in the WebLogic Server documentation.

2. To Compile the Application into an EAR File

To package your application as an EAR file, select **File > Export > EAR file**. In the **EAR Application** field select the name of the EAR project you wish to archive. The EAR archive produced will contain all of the projects that are already components of the selected EAR.

You can also export the build process for projects to an Ant script. For more information see [Creating Custom Ant Build Files for an Application](#).

3. To Deploy the EAR File to the Production Server

Open the WebLogic Server console for the target server running in production mode.

1. On the left-hand side **Change Center** pane, click **Lock & Edit**.
- 2.

On the left-hand side **Domain Structure** pane, click **Deployments**.

3. On the right-hand content pane, click **Install**.
Browse to the directory where the EAR file resides, mark the radio button next to the EAR file you want to deploy, and click **Next**.
4. Confirm that the radio button next to **Install this deployment as an application** is marked and click **Next**.
5. Click **Finish**.
6. On the left-hand side **Change Center** pane, click **Activate Changes**.
7. On the right-hand content pane, mark the radio button next to the EAR just deployed.
8. Click **Start** to view the dropdown list and select **Servicing all requests**.
9. In the content pane of the new page, click **Yes**.

If you click the plus sign in front of the deployed EAR, a list of the application's deployed modules (EJBs and web applications) will appear.

For more information about deploying an application to a server in production mode, see [Deploying Applications on BEA WebLogic Server 9.2](#) in the WebLogic Server documentation.

For information on updating an already deployed application, see [Updating Applications in a Production Environment](#).

Related Topics

[Managing Servers](#)

[Creating a Workshop for WebLogic-Enabled Server Domain](#)

[Setting up Servers for Use Within the IDE](#)

Creating a Workshop for WebLogic-Enabled Server Domain

This topic explains how to create a WebLogic Server domain that can support an application build with Workshop for WebLogic. The instructions below will show you how to add the Apache Beehive NetUI and Controls framework libraries, the JMS queues required for message buffering, reliable messaging, and JMS transport to a server domain.

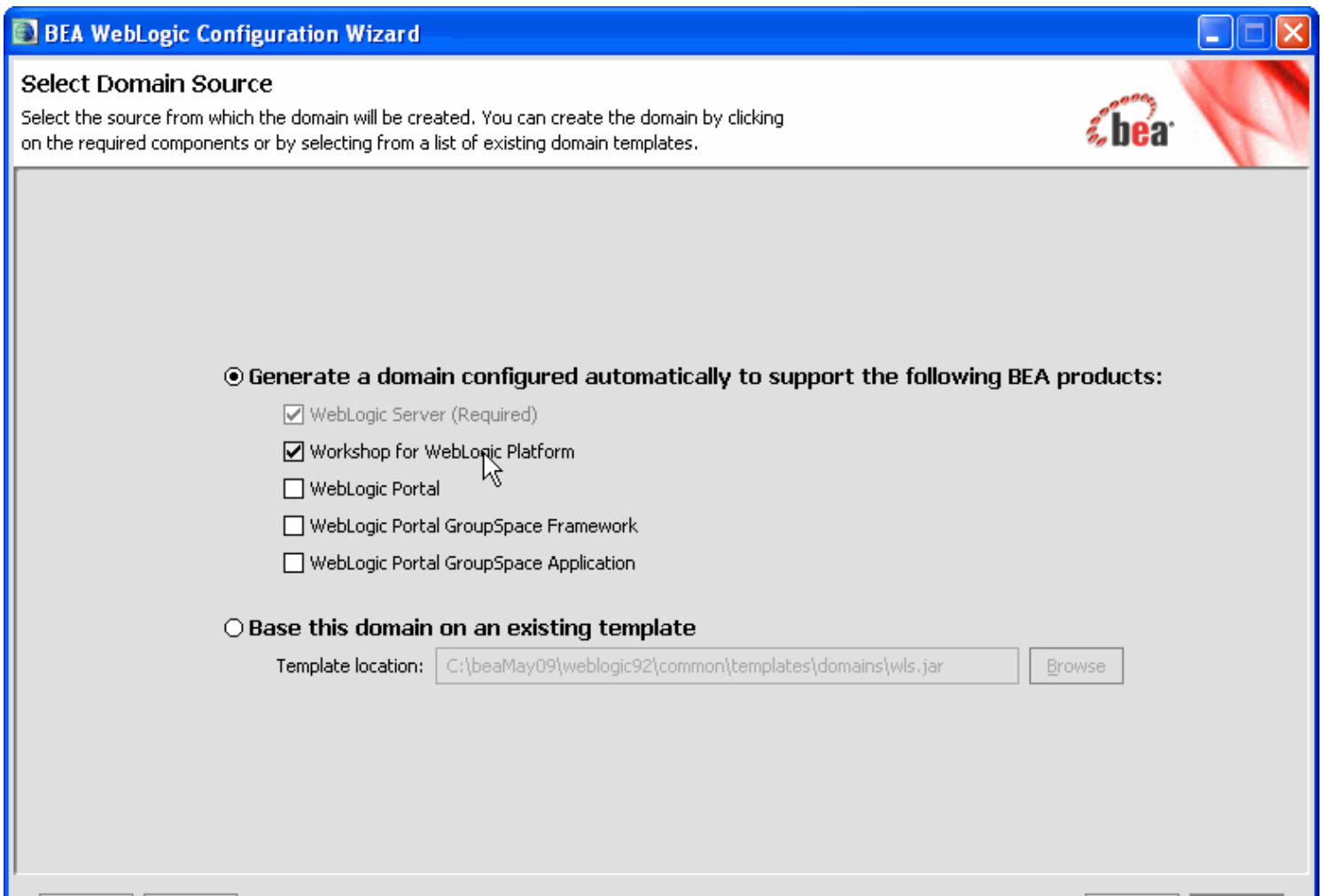
This topic describes two cases for adding Workshop for WebLogic application resources to a server domain:

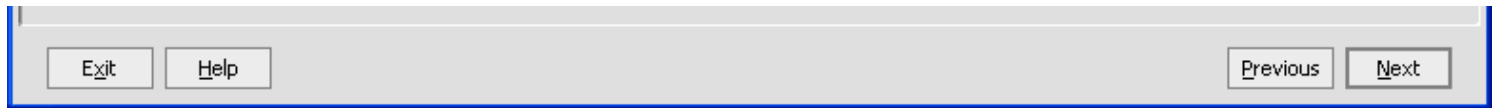
1. [Adding Workshop for WebLogic Resources to a New Server Domain](#)
2. [Adding Workshop for WebLogic Resources to an Existing Server Domain](#)

Adding Workshop for WebLogic Resources to a New Server Domain

If you are creating a new server domain, use these instructions to ensure that the appropriate Workshop for WebLogic resources are present:

1. If you are using Windows, open the domain configuration wizard by selecting **Start > BEA Products > Tools > Configuration Wizard**.
If you are using Linux, open the domain configuration wizard by running **BEA_HOME/weblogic92/common/bin/config.sh**.
2. In the **BEA WebLogic Configuration Wizard**, on the page labeled **Welcome**, confirm that **Create a new WebLogic domain** is selected. Click **Next**.
3. On the page labeled **Select Domain Source**, place a checkmark next to **Workshop for WebLogic Platform**. Click **Next**.
(From a Workshop for WebLogic perspective, this is the important part of the configuration wizard. By placing a check next to Workshop for WebLogic Platform, you ensure sure that domain is configured with the appropriate Beehive libraries, JMS queues, and JDBC datasources.)





4. Complete the rest of the domain configuration wizard as appropriate.

Adding Workshop for WebLogic Resources to an Existing Server Domain

If you are updating a previously existing domain, use these instructions to ensure that the appropriate Workshop for WebLogic resources are present.

1. If you are using Windows, open the domain configuration wizard by selecting **Start > BEA Products > Tools > Configuration Wizard**.
If you are using Linux, open the domain configuration wizard by running **BEA_HOME/weblogic92/common/bin/config.sh**.
2. In the **BEA WebLogic Configuration Wizard**, on the page labeled **Welcome**, select **Extend and existing WebLogic domain**. Click **Next**.
3. On the page labeled **Select WebLogic Domain Directory**, navigate to the root folder of your server domain. Click **Next**.
4. On the page labeled **Select Extension Source**, place a checkmark next to **Extend my domain using an existing extension template**.
Browse to the extension template at `BEA_HOME/weblogic92/common/templates/applications/workshop_wl.jar`. Click **Next**.
(From a Workshop for WebLogic perspective, this is the important part of the configuration wizard. By selecting the extension template, you ensure sure that domain is configured with the appropriate Beehive libraries, JMS queues, and JDBC datasources.)
5. Complete the rest of the domain configuration wizard as appropriate.

For more information on completing the configuration wizard, see [Creating WebLogic Domains Using the Configuration Wizard](#) in the WebLogic Server documentation.

Related Topics

[Deploying, Running, and Debugging Applications](#)

[Managing Servers](#)

[Setting up Servers for Use Within the IDE](#)

Setting Up Logging

Actions, status, and errors within your application are tracked by WebLogic Server log messages. The following topics describes how to turn on message logging for your Workshop for WebLogic applications.

You can configure logging settings by either (1) using the WebLogic Server Administration Console, (2) editing the domain configuration file directly, or (3) through the WebLogic Scripting Tool (WLST).

Configuring Logging with the Administration Console

To open the Administration Console for a given domain, there must already be a running server instance within that domain. To open the Administration Console, either visit <http://localhost:7001/console> or double-click a server on the **Servers** tab and then click the link **Open WebLogic Server Admin Console**.

Follow the instructions at [Create log filters](#) in the WebLogic Server documentation to add a logging filter to the domain. For example, the following logging filter would turn on logging for all packages beginning with "com.bea" and "org.apache.beehive"

```
(SUBSYSTEM LIKE 'com.bea.%') OR (SUBSYSTEM LIKE 'org.apache.beehive.%')
```

Configuring Logging Manually with config.xml

To add a logging filter manually, edit the domain's config.xml file. The config.xml file resides at `<domain_home>/config/config.xml`.

For example, adding the following XML snippet to the config.xml file (added as a child of the `<domain>` element), is equivalent to adding the filter expression `(SUBSYSTEM LIKE 'com.bea.%') OR (SUBSYSTEM LIKE 'org.apache.beehive.%')` using the Administration Console (as described above). When you edit the config.xml file manually, you must restart the server for the changes to take effect.

```
<log-filter>
  <name>beehive_logFilt</name>
  <filter-expression>(SUBSYSTEM LIKE 'com.bea.%') OR (SUBSYSTEM LIKE 'org.apache.beehive.%')</filter-expression>
</log-filter>
```

Configuring Logging with the WebLogic Scripting Tool (WLST)

For information on using the WLST to configure logging, see [Automating WebLogic Server Administration Tasks](#) in the WebLogic Server documentation.

Related Topics

WebLogic Server documentation: [Configuring WebLogic Logging Services](#)

WebLogic Server documentation: [Create log filters](#)

WebLogic Server documentation: [Automating WebLogic Server Administration Tasks](#)

[Managing Servers](#)

Working with Source Control

Projects in Workshop for WebLogic can be stored in source control systems to be shared with other users.

Sharing Project Files with CVS or Another Integrated Source Control System

Eclipse has integrated features for working with CVS repositories. You may also install a plugin that integrates your source control system with Eclipse.

To share a project through CVS or another integrated source control system, right click on the project in the **Page Explorer** or **Navigator** view and choose **Team > Share Project**. You can then use the standard Eclipse commands for storing the project in a source control repository.

For more information, click **Help > Help Contents** and click on the **Workbench User Guide** for more information about using CVS with Eclipse.

Sharing Files with other Source Control Systems

If your source control system is not integrated with Eclipse, you can share a workspace by checking in the workspace directory and its contents **excluding** these directories from checkin. (Not all of these directories and files will exist in every project.)

- .metadata (workspace-level)
- build (project-level)
- templib (project-level)
- .apt_src (project-level)
- .xbean_src and .xbean_bin (project-level; only for projects with XMLBeans Builder enabled)

The following directories and files should be **included** in source control. (Not all of these directories and files will exist in every project.)

- .settings/* (project-level)
- .classpath (project-level)
- .factorypath (project-level)
- .project (project-level)

- .datasync-project (project-level)

Making a Portable Workspace ZIP File

You can also create a ZIP file that contains the workspace and projects.

If you are making a ZIP file manually or through an Ant task, make sure to **exclude** these directories from the ZIP file:

- .metadata (workspace-level)
- build (project-level)
- templib (project-level)
- .apt_src (project-level)
- .xbean_src and .xbean_bin (project-level; only for projects with XMLBeans Builder enabled)

If you are making a portable ZIP file with Workshop for WebLogic, select **File > Export > Archive file > Next**. In the left-hand pane, select the projects you want to include, but *unselect* the following directories within each project:

- build
- templib
- .apt_src
- .xbean_src and .xbean_bin (only for projects with XMLBeans Builder enabled)

To retain the original directory structure when your workspace has multiple projects, make sure to place a checkmark next to **Create directory structure for files**.

To uncompress the ZIP file and use the workspace, follow the instructions in [Workshop for WebLogic Samples](#).

Other Issues when using Source Control

When projects are to be shared using source control, you may want to [save your source files outside the workspace](#).

Related Topics

[Applications and Projects](#)

Exporting Archive Files

Your Workshop for WebLogic projects can be exported into archive files:

Project Type	Archive File
Enterprise Application Project	Enterprise ARchive (EAR)
Web project (Web Service Project, Dynamic Web Project)	Web ARchive (WAR)
Utility Project	Java ARchive (JAR)
EJB Project	EJB Java ARchive (EJB JAR)

You can:

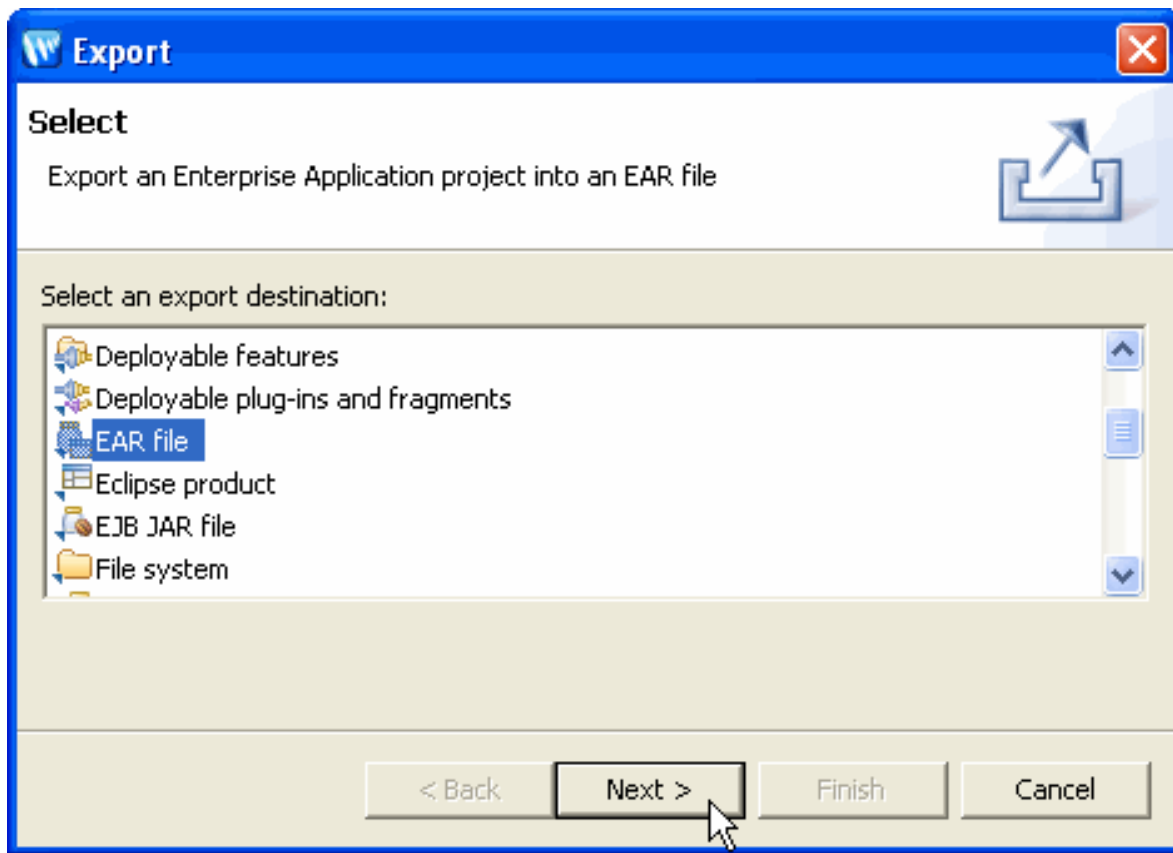
- [Export an Enterprise Application Project to an EAR](#)
- [Export a Web Project to a WAR](#)
- [Export a Utility Project to a JAR](#)
- [Export an EJB Project to an EJB JAR](#)
- [Export Controls in a Utility Project to a Control JAR](#)

Note that options on the **Export** dialog that are not described in this documentation do not apply to Workshop for WebLogic projects. For example, you can use the **Archive** and **JAR file** options of the **Export** dialog to create JAR files for EJB and web projects. However this is only a useful feature for creating utility JAR files for sharing code.

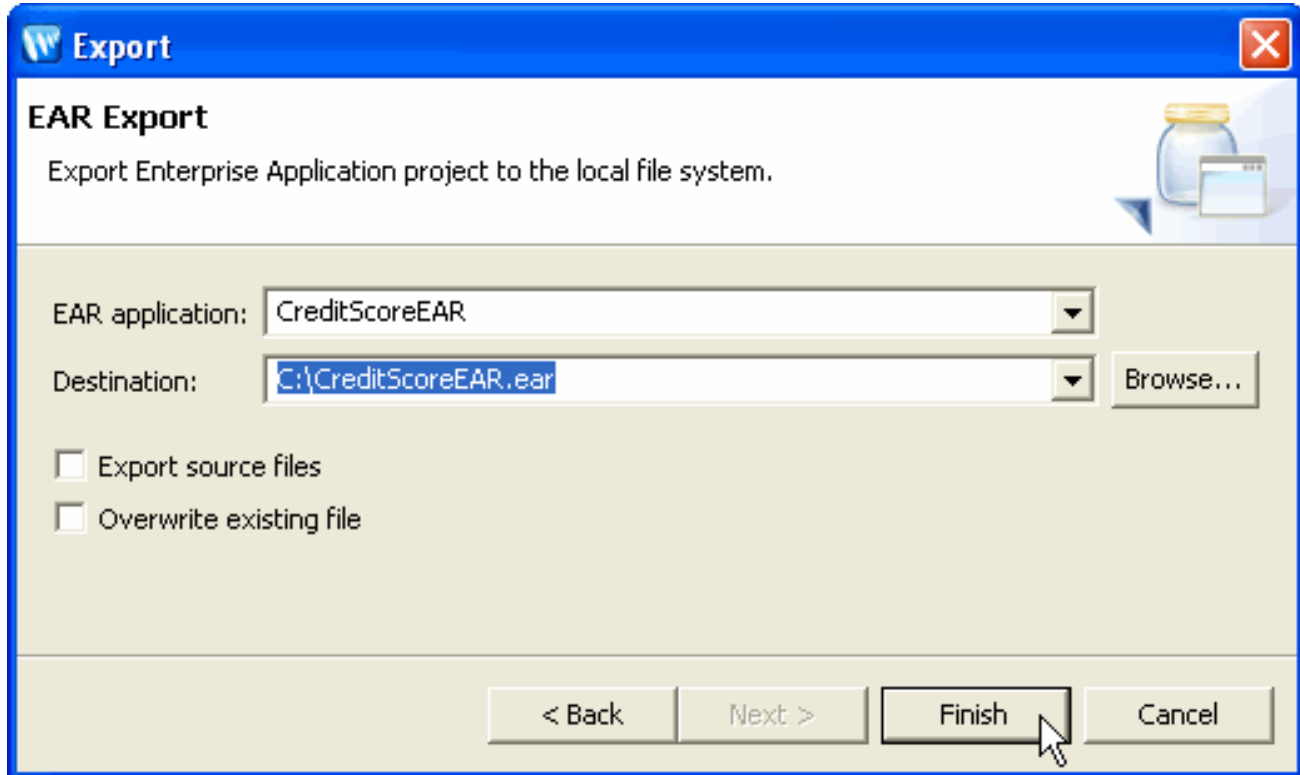
How to Export an Enterprise Application Project to an EAR

To export an EAR project to an EAR file:

1. Right click anywhere on the **Package Explorer** view and choose **Export** from the menu.
2. Click **EAR file** and click **Next**.



3. Use the pulldown to select the EAR project in the **EAR application** box. Use the **Browse** button to specify the location where the new EAR file will be located.



If the file already exists, you must check the **Overwrite existing file** selection to replace the existing file.

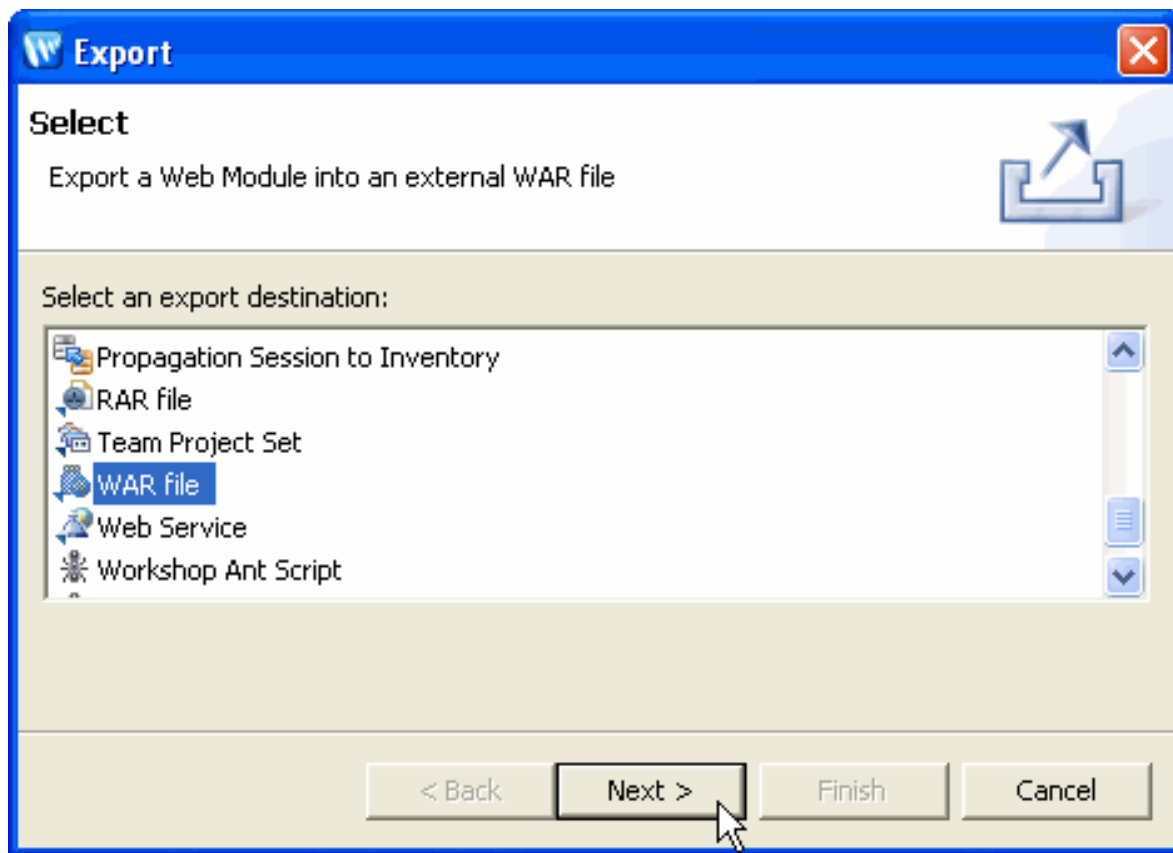
Normally the **Export source files** option is unchecked. If you wish to include the Java source files in your archive, you must check this option. The most common reason to export an EAR file is to deploy for production, and in that case, the source files should not be included. Include the source files only if you wish to view/edit source at the destination.

4. Click **Finish** to create the EAR file.

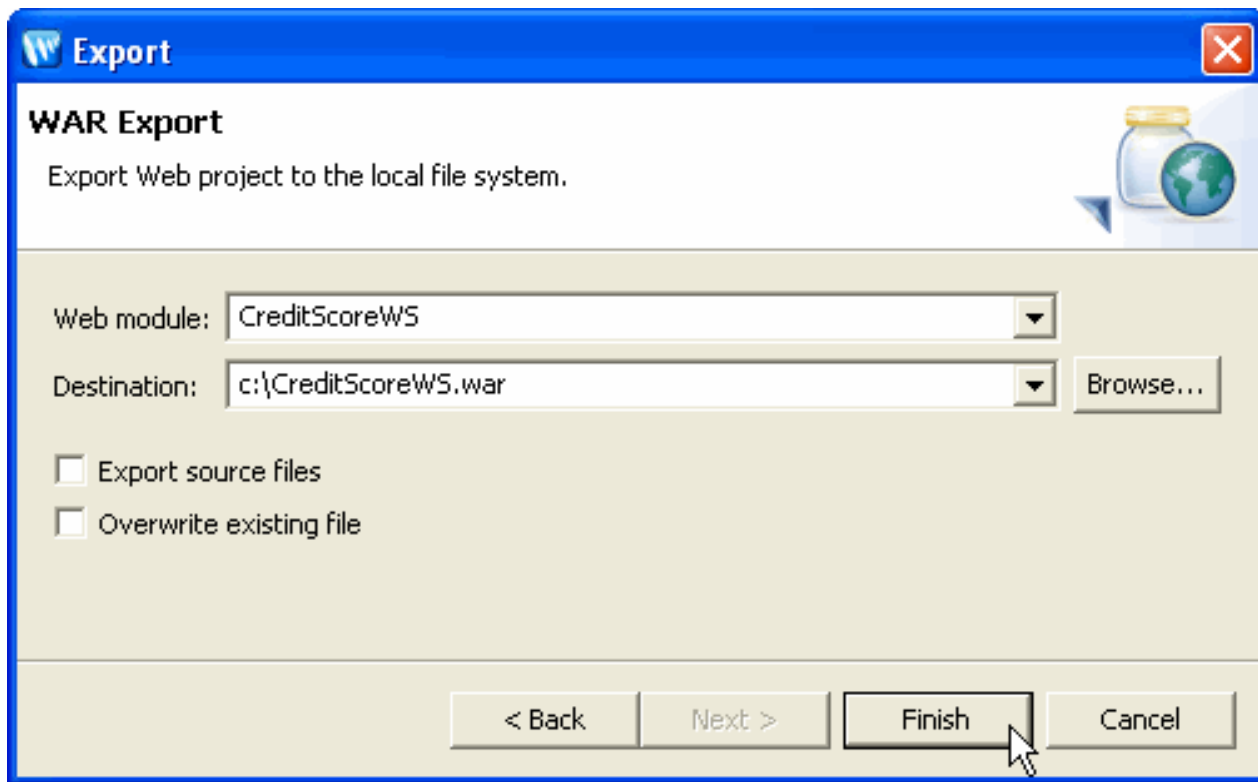
How to Export a Web Project to a WAR

To export a web project to a WAR file:

1. Right click anywhere on the **Package Explorer** view and choose **Export** from the menu.
2. Click **WAR file** and click **Next**.



3. Use the pulldown to select the web project in the **Web module** box. Use the **Browse** button to specify the location of the WAR file.



If the file already exists, you must check the **Overwrite existing file** selection to replace the existing file.

Normally the **Export source files** option is unchecked. If you wish to include the Java source files in your archive, you must check this option. The most common reason to export a WAR file is to deploy for production, and in that case, the source files should not be included. Include the source files only if you wish to view/edit source at the destination.

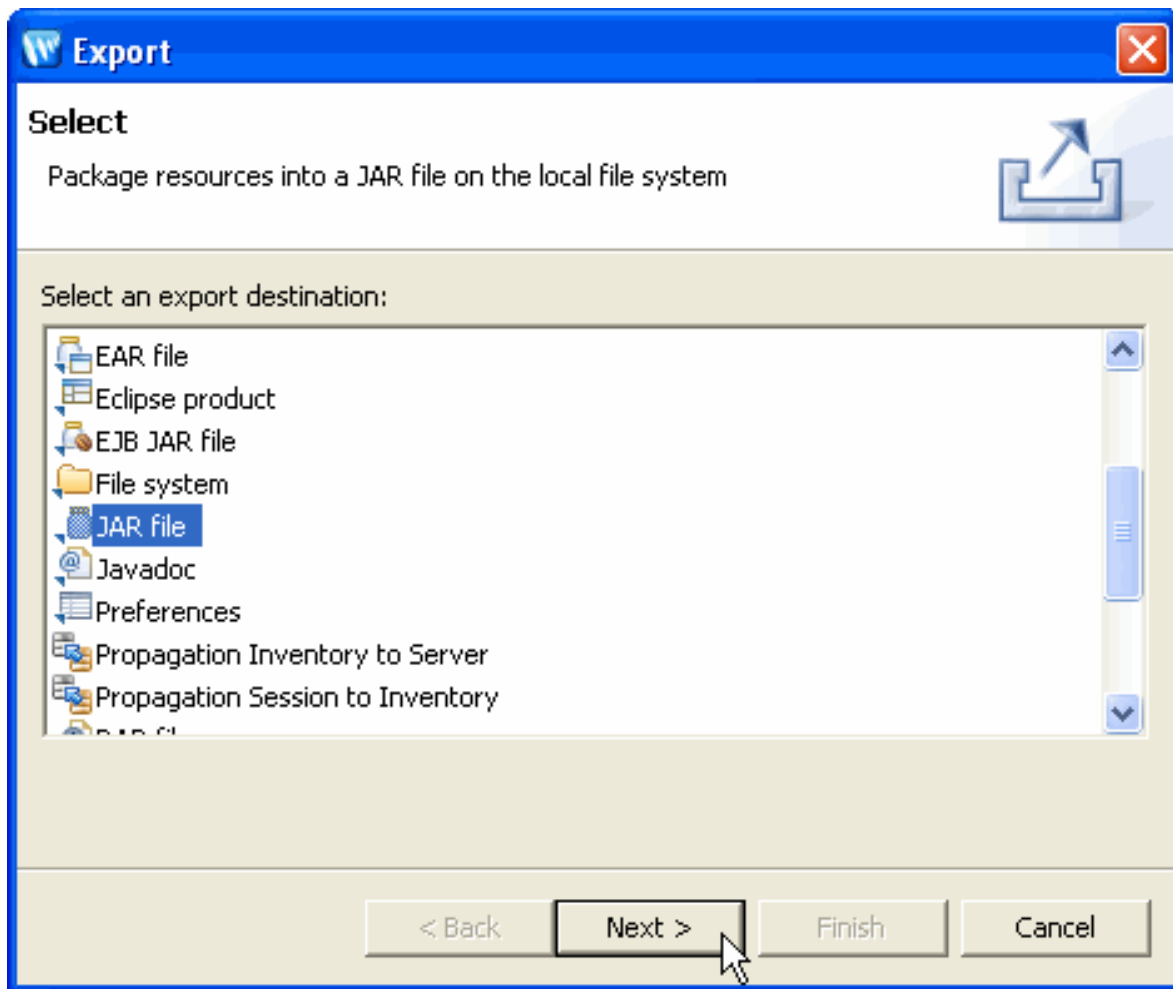
4. Click **Finish** to create the WAR file.

How to Export a Utility Project to a JAR

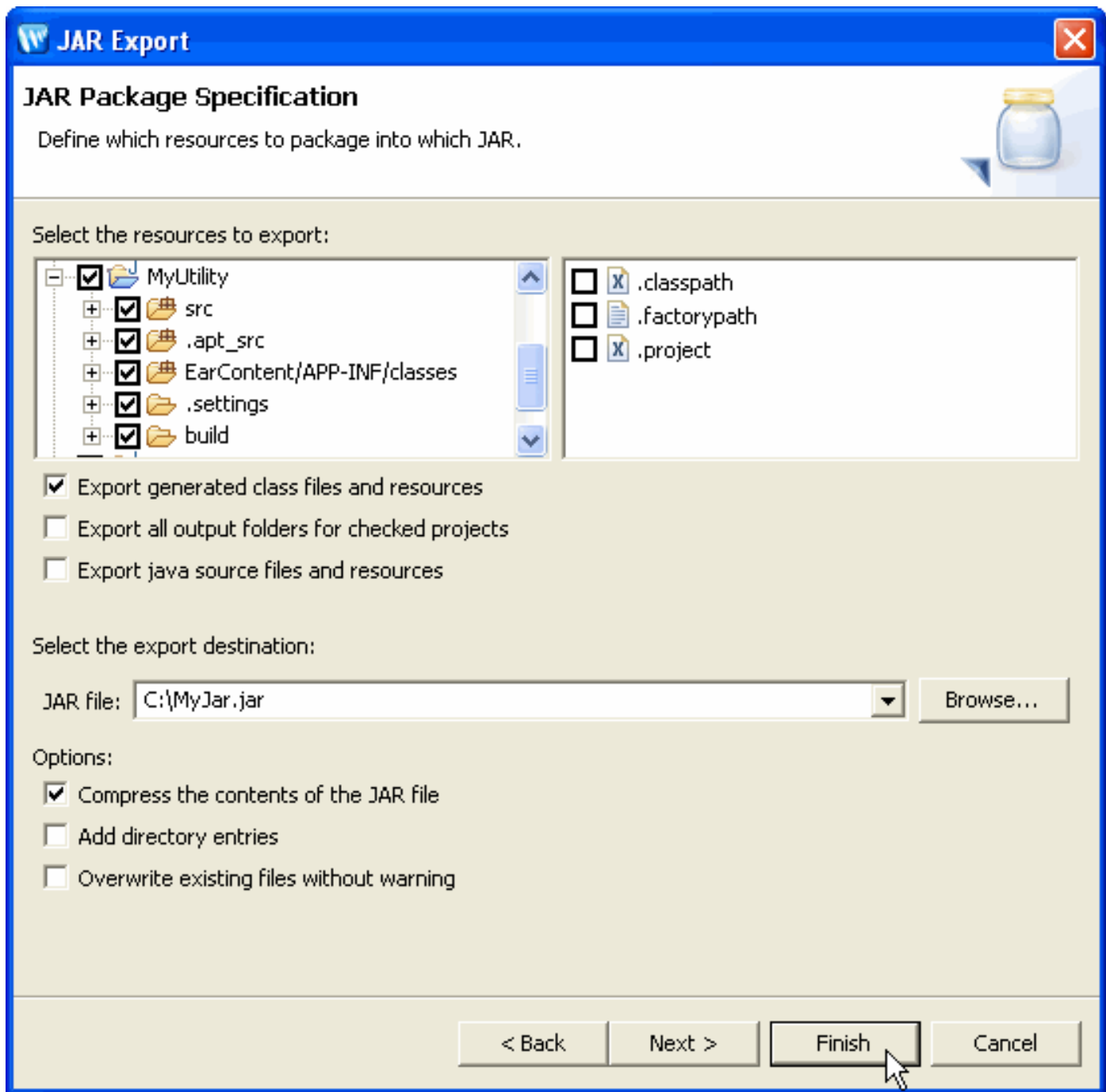
Note that if you use this process to create a JAR file from a utility project, then the controls within the JAR will not appear in the IDE correctly (although they will work correctly). You can also [export the controls in a utility project to a control JAR](#).

To export a utility project to a JAR file:

1. Right click on the name of the utility project in the **Package Explorer** view and choose **Export** from the menu.
2. Click **JAR file** and click **Next**.



3. Click the checkbox to select the utility project that you wish to export. Use the **Browse** button to specify the location of the JAR file that will be created.



If the file already exists, you may check the **Overwrite existing files without warning** selection to replace the existing file.

Normally the **Export source files** option is unchecked. If you wish to include the Java source files in your archive, you must check this option. The most common reason to export an EAR file is to deploy for production, and in that case, the source files should not be included. Include the source files only if you wish to view/edit source at the destination.

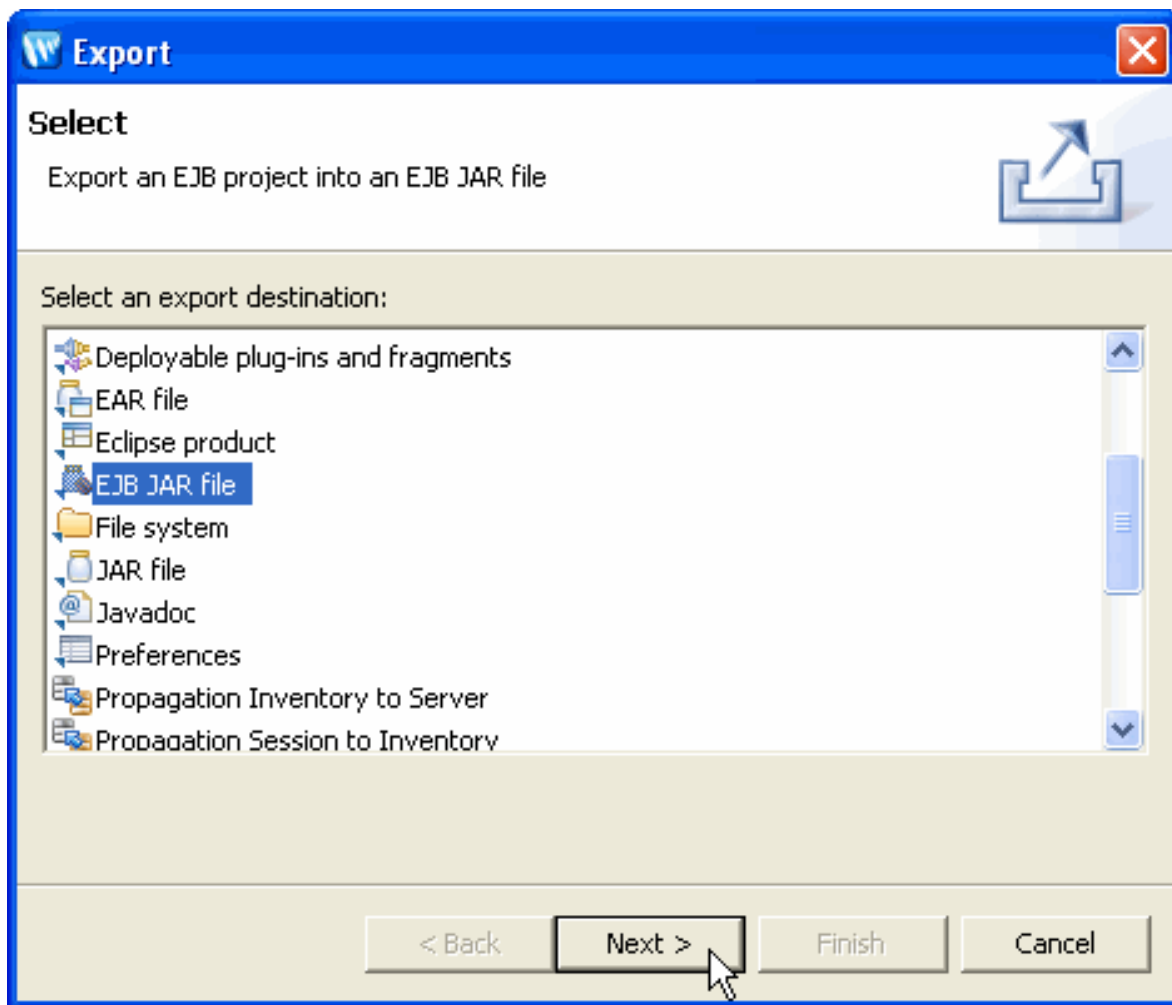
The screens that manage the JAR format are standard Eclipse screens documented in the **Workbench User Guide**.

4. Click **Finish** to create the JAR file.

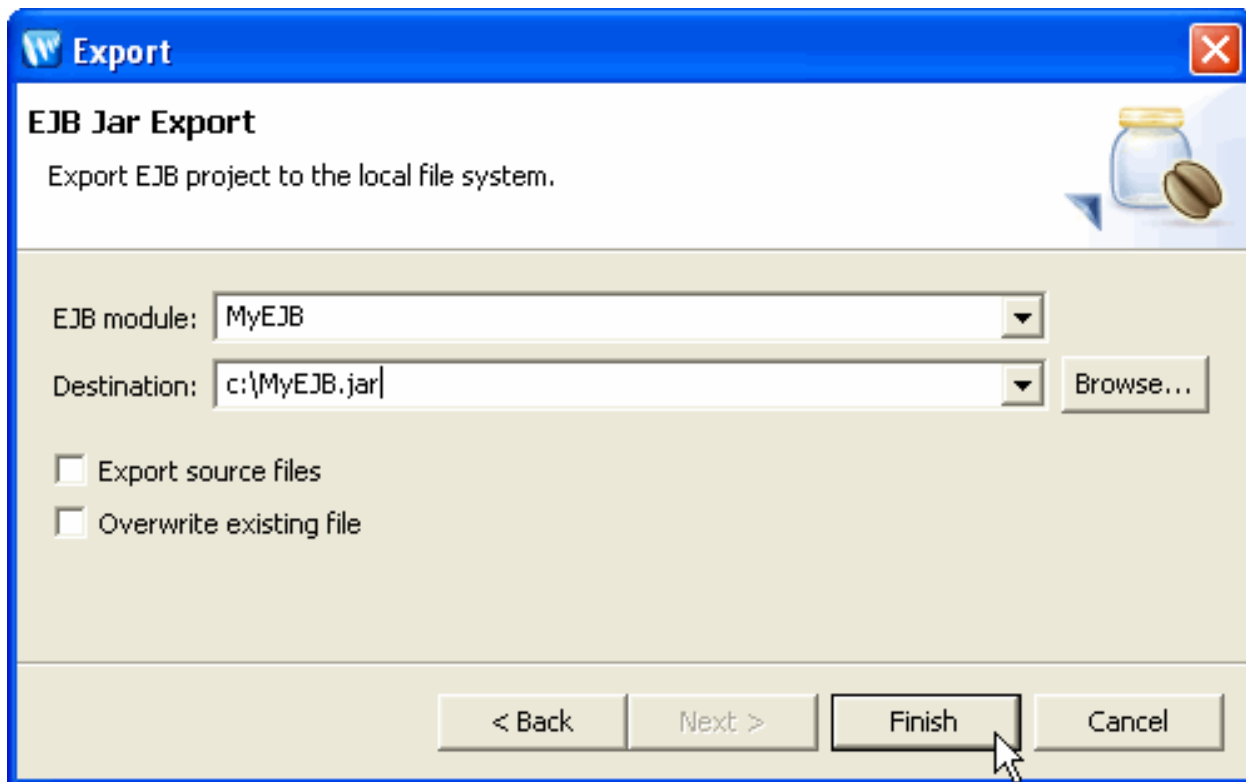
How to Export an EJB Project to an EJB JAR

To export an EJB project to an EJB JAR file:

1. Right click anywhere on the **Package Explorer** view and choose **Export** from the menu.
2. Click **EJB JAR file** and click **Next**.



3. Use the pulldown to select the EJB project in the **EJB module** box. Use the **Browse** button to specify the location of the new EJB JAR file.



If the file already exists, you must check the **Overwrite existing file** selection to replace the existing file.

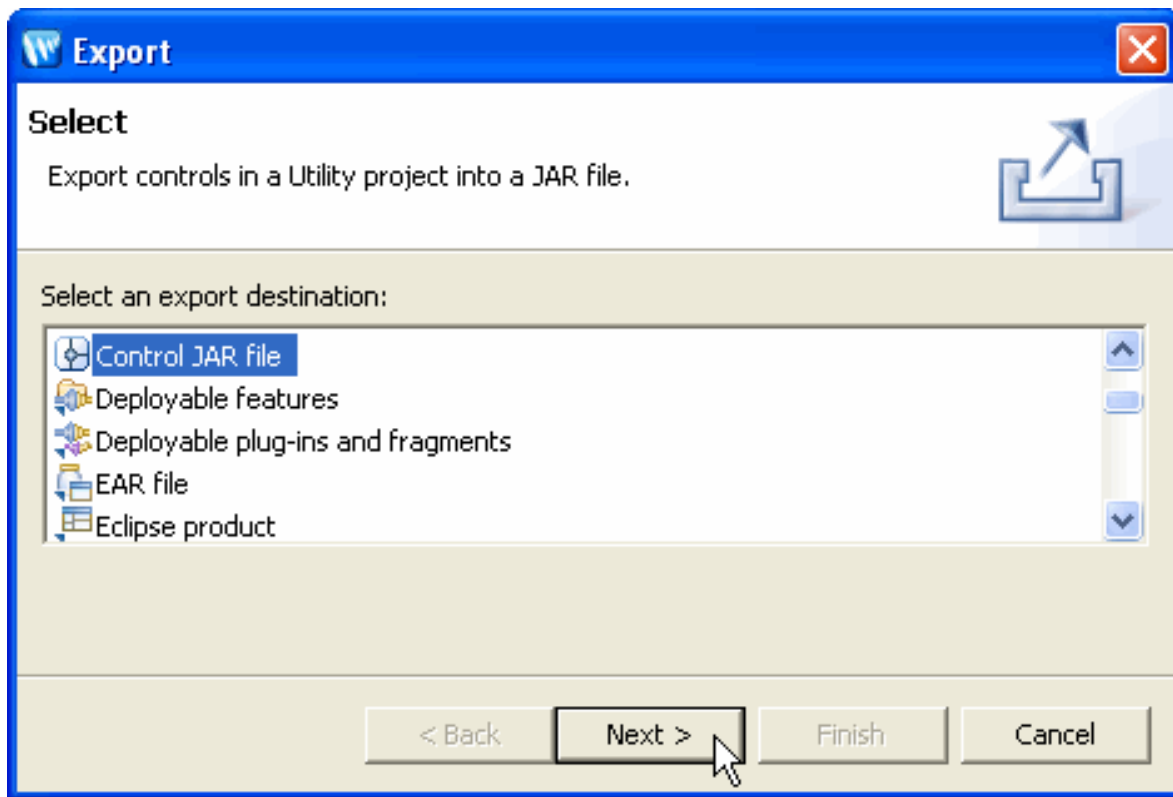
Normally the **Export source files** option is unchecked. If you wish to include the Java source files in your archive, you must check this option. The most common reason to export an EJB JAR file is to deploy for production, and in that case, the source files should not be included. Include the source files only if you wish to view/edit source at the destination.

4. Click **Finish** to create the EJB JAR file.

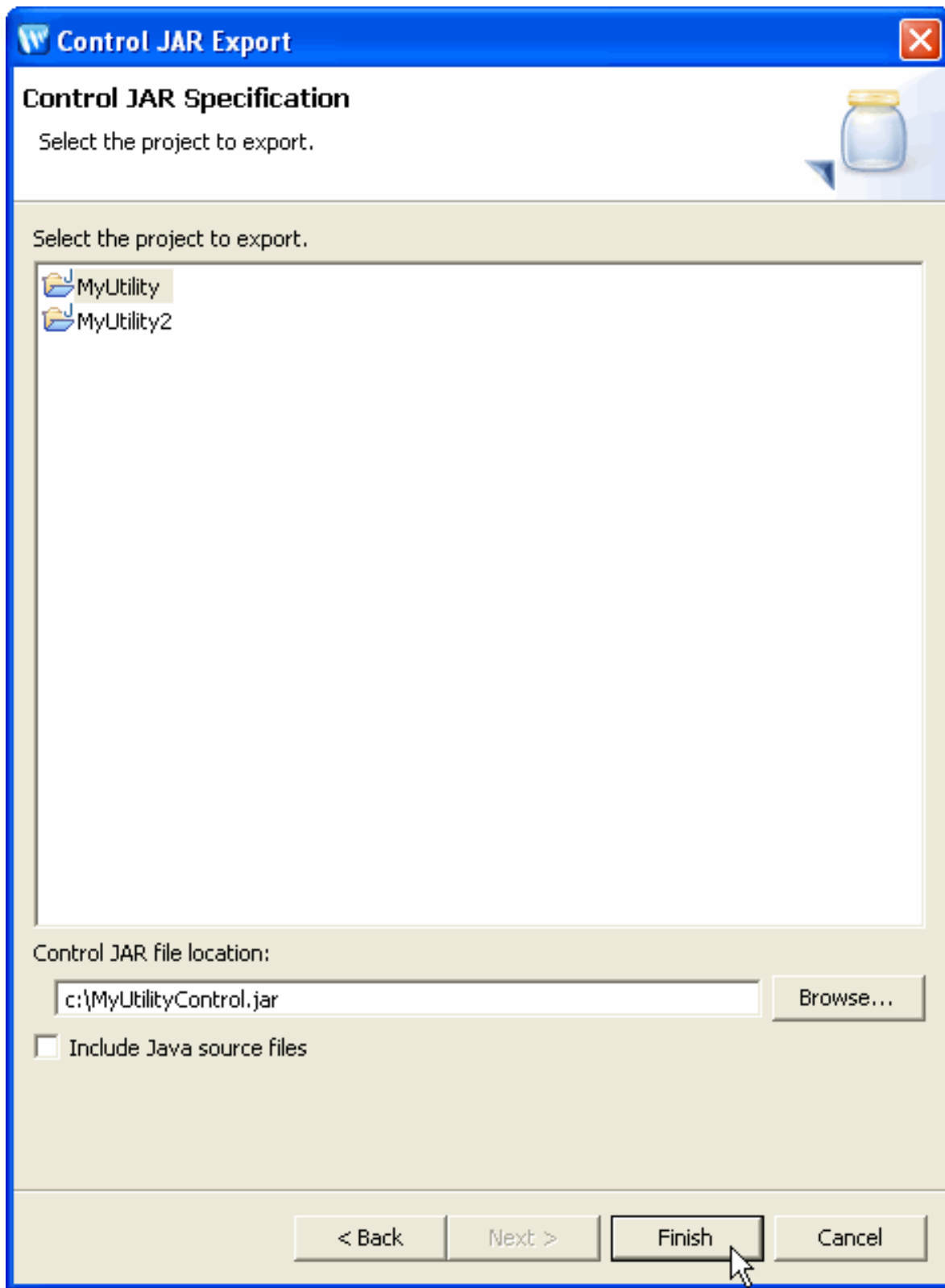
How to Export a Utility Project with Beehive Controls to a JAR

To export the controls in a utility project to a JAR file, follow the steps below. The resulting control JAR contains all of the contents of the utility project, but also stores the Beehive control manifest entries so that controls are displayed correctly in the IDE.

1. Right click anywhere on the **Package Explorer** view and choose **Export** from the menu.
2. Click **Control JAR file** and click **Next**.



3. Highlight the name of the utility project to use for the export. Use the **Browse** button to specify the location of the JAR file that will be generated.



Normally the **Include Java source files** option is unchecked. If you wish to include the Java source files in your archive, you must check this option. The most common reason to export a JAR file is to deploy for production, and in that case, the source files should not be included. Include the source files only if you wish to view/edit source at the destination.

4. Click **Finish** to create the control JAR file.

Related Topics

none.

Creating Custom Ant Build Files for an Application

If you want to extend or modify the standard build process, Workshop for WebLogic will export an [Ant script](#) that duplicates the standard application build. You can then modify the Ant script as needed and do command line builds. Modifying the generated Ant script does not alter the build process within Workshop for WebLogic.

Ant Build Script Functionality

Workshop for WebLogic generates the following types of Ant build scripts:

- **Web Project** (generated for Dynamic Web Projects)
- **EAR Project** (generated for Enterprise Application Projects)
- **Utility Project** (generated for Utility Projects)
- **EJB Project** (generated for WebLogic EJB projects)

The exported Ant script contains a set of standard targets for building, assembling, staging and generating module archives. You can view the targets supported for a specific script with the command `ant -projecthelp`. The functionality of these targets is detailed in the following table.

Supported Ant Targets

Target	Dependencies	Details
build		Compiles the source files; does not package the results. The target in an EAR project's Ant script builds all child module projects in dependency order.
assemble	<i>build</i> (a formal dependency does not exist on "build" but that target must be called prior to assemble)	Assembles the project for iterative dev deployment; requires that the "ear.root" property is specified. The target in an EAR project's Ant script assembles all child module projects.
stage	<i>build</i> (a formal dependency does not exist on "build" but that target must be called prior to stage)	Copies all of project's resources and build artifacts into a form that's ready for archive creation; staging directory can be overridden via the "staging.dir" property. The target in an EAR project's Ant script stages all child module projects.

archive	stage	Creates an archive containing all projects's resources and build artifacts; archive name and location can be overridden via the "archive.path" property. Note that the archive target builds an EAR file if run from an EAR script; if run from an Ant script generated from a web project, it generates a WAR file.
clean		Removes the files and directories generated by the build target

Generating Ant Build Scripts

To Export an Ant Build Script for a Project:

1. Right click on the project name in the **Package Explorer** view and choose **Export** to display the **Select** dialog.
2. Choose **Workshop Ant Script**. Click **Next**.
3. In the **Ant Script Generators:** pane, click on the type of Ant script to generate:
 - o For web projects, choose **Web Project Build Script (WebLogic Server)**
 - o For EAR projects, choose **Ear Project Build Script (WebLogic Server)**
 - o For utility projects, choose **Utility Project Build Script (WebLogic Server)**
 - o For WebLogic EJB projects, choose **EJB Project Build Script (WebLogic Server)**
 - o For all other project types (i.e. WTP EJB, Connector, Application Client, Java), only the **Java Project Build Script** is available. This build script provides minimal functionality (just basic build and clean targets) and cannot be used directly within the context of a full EAR build. For these projects, you may find it useful to instead customize the build script created for a utility project.
4. In the **File Name** box, choose the path and file name for the generated Ant script. Click **Finish**.

To Export an Ant Build Script for an Application:

To generate Ant scripts for an entire application, right click on each project name and generate the Ant script for each member project (web, utility, and EJB projects) individually. Then right click on the EAR project and generate the Ant script.

To Generate the Workspace Metadata File:

If you want to run your Ant scripts on a remote machine that does not have access to the workspace, you must first generate a workspace metadata file:

1. Choose **File > Export** to display the **Select** dialog.
2. Choose **Workspace Metadata for Workshop Ant Script** . Click **Next**.
3. From the **Select Projects** dialog:
 - o On the **Projects** pane, choose the projects to use when creating the workspace metadata file.
 - o In the **Export Destination** box specify the location and name for the workspace metadata file.

Click **Next** to continue.

4. On the final screen, The `wl.home` and `workspace.dir` will be automatically populated. The purpose of that screen is to define variables to relativize absolute paths that would be written into the metadata file. The user can see the remaining absolute paths in the lower box. When a new variable is defined that covers a portion of one of the absolute paths, those absolute paths will be removed from that box. Every variable that's defined at this point will need to be passed to the ant script at runtime using the `-D` syntax. Click **Finish**.

Caveats and Implementation Notes:

- Eclipse working sets are not supported.
- Projects must be open (**Project > Open Project**) before exporting an Ant script. You cannot export an Ant script for a closed project.
- The Ant script for a project should be regenerated if any changes are made to the project's enabled facets.
- The workspace metadata file must be regenerated if changes are made to project metadata (e.g. Java build path) or workspace structure (e.g. new/removed projects, changes in inter-project dependencies).

Executing Ant Build Scripts

To Build a Project by Running the Ant Script:

1. Generate the Ant script for the project.
2. Configure the execution environment of your shell by executing `wl.home/common/bin/commEnv.cmd` (or `wl.home/common/bin/commEnv.sh` for Linux).

Use of the Ant scripts without prior execution of `commEnv.cmd` (`commEnv.sh` for Linux) is not currently supported.

3. Change to the Eclipse project directory.
4. Execute the desired Ant targets as follows (example illustrates the creation of the project archive):

```
ant build archive -Dworkspace=workspacepath
```

where the parameters are:

workspacepath is the full path for the workspace folder (e.g., on Windows you might specify the path as C:/MyWork/Workspaces/MyApp) OR the full path and file name of the workspace metadata file.

If your path names have embedded spaces, you must use quote marks to delimit the path name, e.g.,

```
ant build -Dworkspace="C:\Documents and Settings\MyWorkspace"
```

To Build an EAR File Using the Ant Script:

1. Generate the Ant script for **each** project in the EAR.
2. Generate the Ant script for the EAR project.
3. Configure the execution environment of your shell by executing *wl.home/common/bin/commEnv.**
4. Change to the Eclipse directory for the EAR project.
5. Execute the desired target in the EAR project's Ant script as follows:

```
ant build archive -Dworkspace=workspacepath
```

To Execute Ant Scripts on a Remote Machine:

1. Generate the workspace metadata file as discussed above.
2. Generate the necessary project Ant build scripts.
3. Copy the following files to the remote machine:
 - o the Ant script(s)
 - o the workspace metadata file
 - o everything else in the project and/or ear file, minus the contents of the build directories
4. Following the project or EAR build instructions above but specify the location of the workspace metadata file as the value of *-Dworkspace=* argument instead of the workspace directory location.

Caveats and Implementation Notes:

- Workspace metadata is imported by the Workshop Ant scripts during every execution. Unless an exported workspace metadata file is being used, this behavior ensures that changes to the workspace metadata (e.g. modifications to a project's classpath) will be reflected without

necessitating a regeneration of the script. The state of the imported workspace metadata can be viewed by specifying `-Decho.metadata=true` on the commandline when executing Ant.

- Generated Ant scripts are for use in command line builds, not to run within Workshop for WebLogic. Concurrent use of Workshop for WebLogic and the Ant scripts is not supported and can lead to build errors. The output from the Ant script build should be cleaned prior to launching Workshop for WebLogic to ensure there are no conflicts.

Related Topics

[Understanding the Build Process](#)

[Apache Ant online Manual](#)

Using XMLBeans in the IDE

Workshop for WebLogic includes XMLBeans, an open source technology for binding XML to Java objects. XMLBeans provides multiple ways to access bound XML, but within Workshop for WebLogic the principal means is by generating Java types by compiling WSDL and XSD (schema) files, then binding XML to the generated types.

This topic includes information on:

[Compiling Schema for Java Types to Use in Projects](#)

[Enabling Automatic Java Type Generation with the XMLBeans Builder](#)

[Turning off Automatic Schema Compilation](#)

[Generating a JAR File that Contains Java Types Generated from Schema](#)

[Locating Generated Java Types](#)

[Guiding XMLBeans Type and Package Naming During Compilation](#)

[Supporting the Interface Extension Mechanism](#)

Note: XMLBeans is one of the Apache open source technologies. For more information and documentation on XMLBeans, see the [Apache XMLBeans web site](#).

Specifically, in Workshop for WebLogic you might find that XMLBeans are handy for the following:

- Use Java types generated from a schema or WSDL as parameter or return types in web service operations. When you use a generated type as a parameter, incoming XML that conforms to that schema type will be automatically bound to the parameter type. For more information on XMLBeans' use in web services, see [Web Service Development Starting Points](#).
- Use generated types to manipulate XML within your code. See [related XMLBeans documentation](#).
- Traverse the full XML infoset using a cursor model. See [related XMLBeans documentation](#).
- Reflect into the structure of the source XML schema through the XML Schema Object model. See [related XMLBeans documentation](#).

Compiling Schemas for Java Types to Use in Projects

If you want to use types generated by XMLBeans from schema in your project, you can do so in one of two ways: with automatic schema compilation through the XMLBeans Builder or by creating a JAR file that contains the generated types.

Important: You should not use both of these techniques with the same schemas in the same project. If you do, you will end up with two sets of essentially the same generated types on your project's classpath. If you begin with one technique, be sure to "undo" the other before switching. For example, if you've added the XMLBeans Builder facet, be sure to remove the facet (or remove the schemas from the [schema source path](#)) before generating a types JAR. Conversely, you should remove a generated types JAR from your project's classpath before adding the XMLBeans Builder facet.

When you remove the XMLBeans Builder facet, the IDE removes Java types that the builder generated.

XMLBeans Builder

You can have the IDE [automatically generate Java types](#) from your WSDL and XSD files by using the XMLBeans Builder. In this way, as you make changes to the WSDL or schema, updated types are generated. (This facet replaces the functionality provided by the Schemas project type in WebLogic Workshop version 8.1.)

The builder is a project facet that:

- Examines [specified directories](#) for WSDLs and schemas.
- Generates Java types by compiling schema types described in the files.
- Puts the generated types on your project classpath. For information about where the generated types go, see [Locating Generated Java Types](#).

Note that you can [create a JAR file from automatically generated types](#).

Types JAR

You can [create a JAR file with the generated types](#), then put the JAR file on your project's classpath. You might prefer this option if you don't anticipate any changes to the WSDL or schema from which types are generated. By using a types JAR rather than the XMLBeans Builder, you can also avoid the overhead present when the IDE compiles schemas at build time. Note that if your WSDL or schema files change, you'll need to manually recreate the JAR file.

Enabling Automatic Java Type Generation with the XMLBeans Builder

You get automatic schema compilation by using the XMLBeans builder, a facet you can add to your project. The XMLBeans builder is an Eclipse incremental project builder (similar to the Java builder) that can be enabled on either WTP faceted projects (via the XMLBeans Builder facet) or on plain Java projects (by selecting the Enable XMLBeans Builder check box in the XMLBeans page of the project's properties).

When you enable the XMLBeans Builder facet, the IDE creates a "schemas" directory in your project (you can rename the directory, add additional directories or specify inclusion/exclusion filters later) and adds this directory to a schema source compilation path. WSDL and XSD files you put into this directory will be automatically compiled (when your project is built) into Java types that the IDE puts on your project's classpath. You can change the schema compilation path as described in [Setting the Schema Compilation Source Path](#). (See the [note above](#) for important information about the XMLBeans Builder.)

For WTP faceted projects (WebLogic EJB, Dynamic Web and Utility), you can add the XMLBeans Builder in one of two ways:

- When you create the project with the New Project wizard, by selecting the XMLBeans Builder facets.
- After you create the project, by using the project properties dialog to add the XMLBeans Builder facets through the Project Facets category.

In either case, the user interface through which you add the facet is pretty much the same. The steps include:

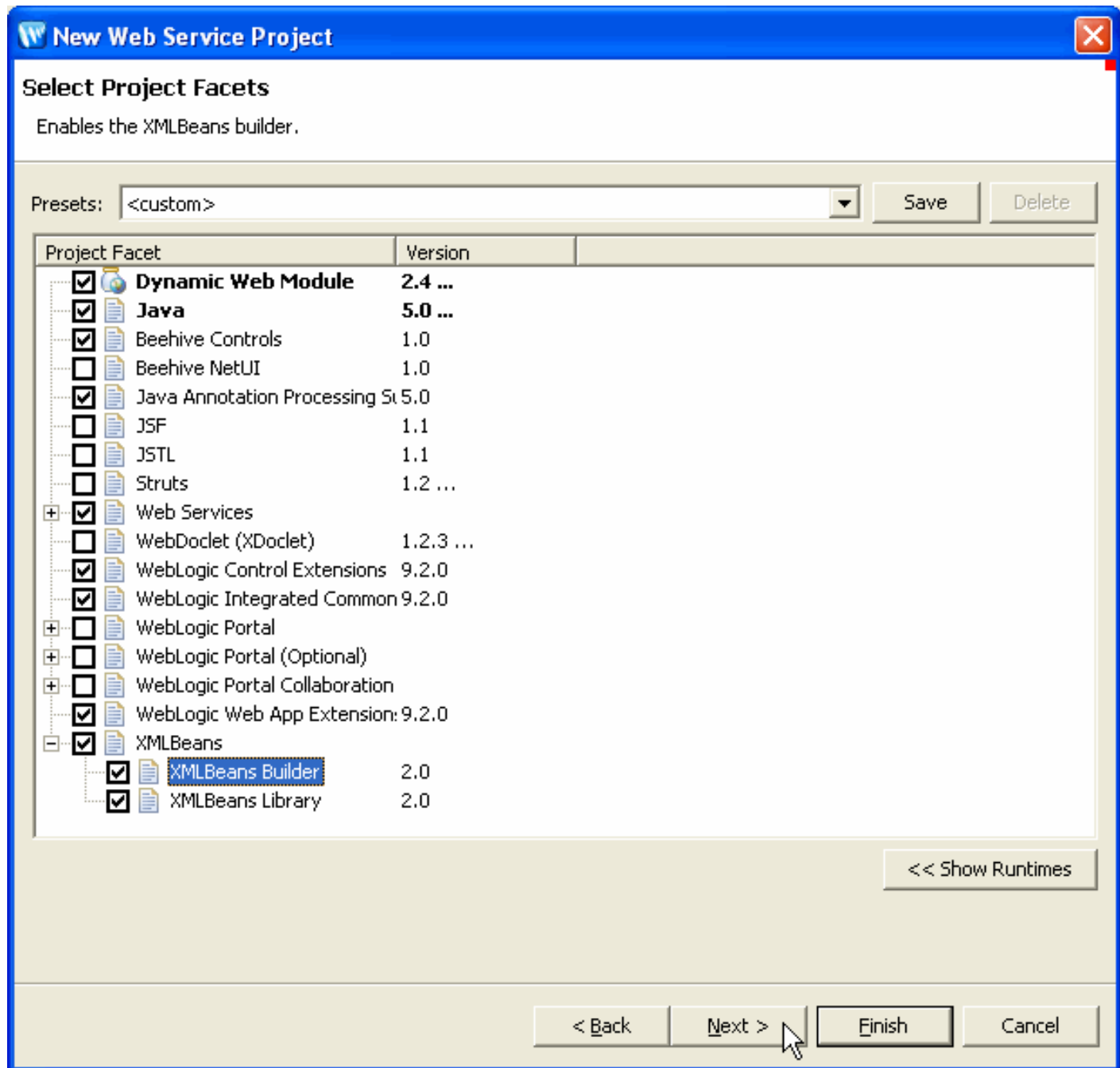
1. [Adding the XMLBeans Builder facet](#).
2. [Setting generated file locations](#).
3. [Setting XMLBeans options](#).
4. [Setting the schema compilation source path](#).

For plain Java projects, you can enable the XMLBeans builder via the XMLBeans page in the project properties dialog.

Adding the XMLBeans Builder Facet

Under Select Project Facets, you select the XMLBeans Builder facet by expanding the XMLBeans node, then selecting the XMLBeans Builder check box. (The following user interface is also used when you add the facet to an existing project.)

For an introduction to project facets, see [Setting Project Facets](#).



The options include:

XMLBeans Library (selected by default for all project types except EAR projects)

- Adds apache_xbean.jar to project classpath. (Will also expose jsr173_api.jar if the WebLogic server runtime is not supported on the project.)

- Required for the XMLBeans Builder and for projects containing XMLBeans-based code.

XMLBeans Builder

- Enables the XMLBeans incremental project builder. This builder executes the XMLBeans compiler incrementally on XSD, WSDL and XSDCONFIG files contained in the [schema compilation source path](#); by default, this path is a "schemas" folder in the project. Java source files on this path are also used as input for interface extension generation.

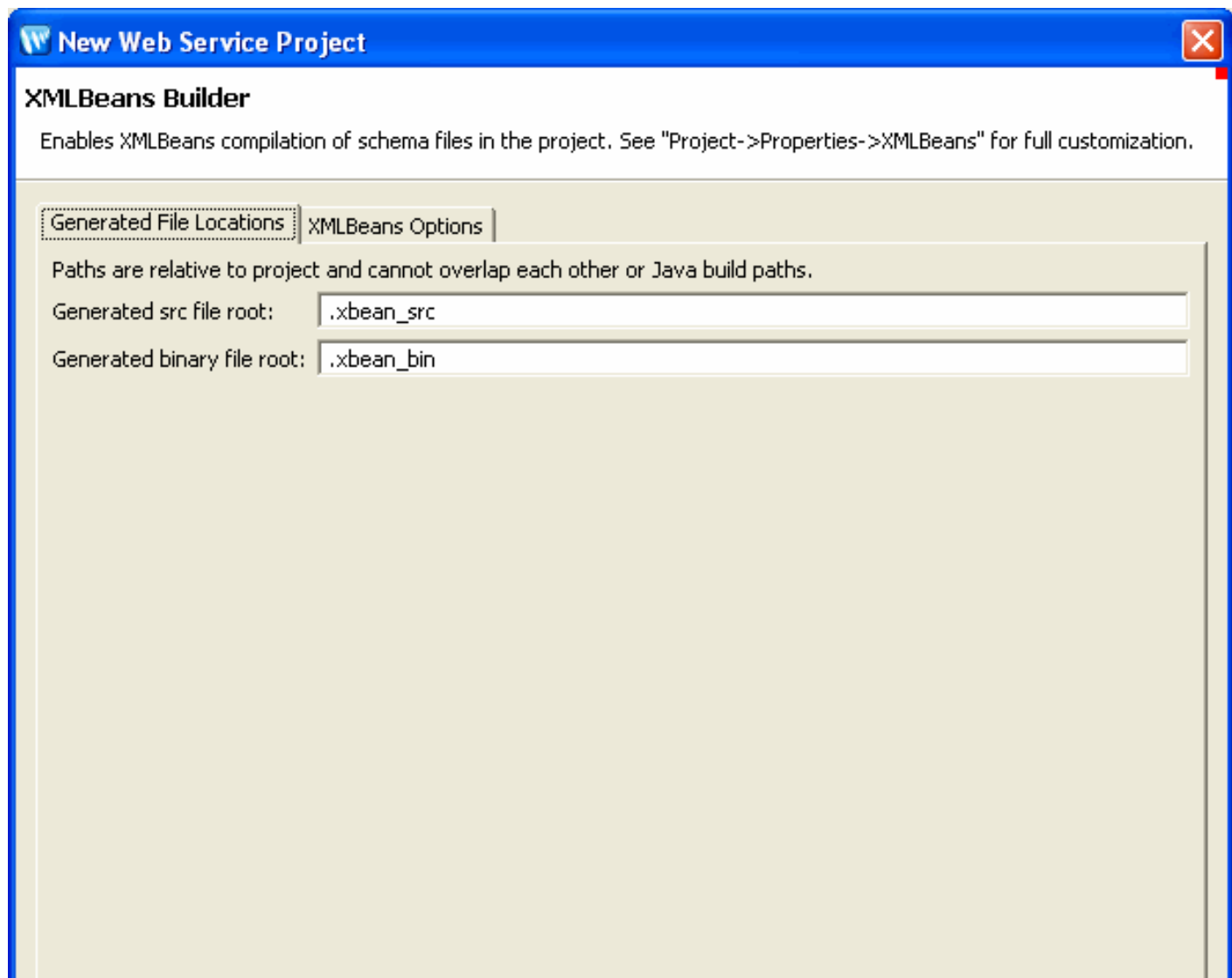
After you've selected the XMLBeans Builder facet, click Next.

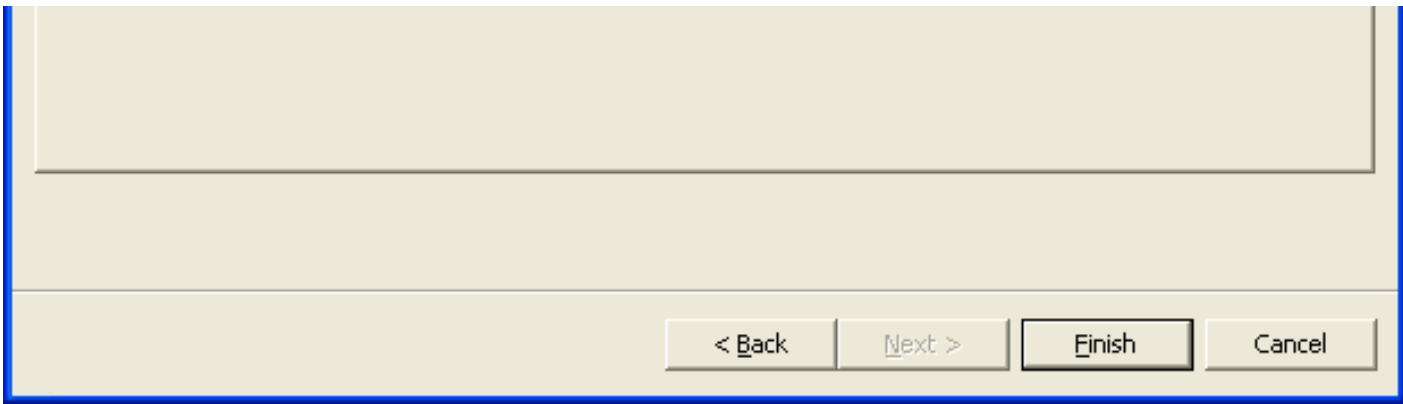
Selecting Builder Options

Once you've selected the XMLBeans Builder facet and clicked Next, you'll find two tabs through which you can set properties related to Java type generation. These include user interface for [setting the locations for source and binary files](#) generated during schema compilation and for [setting XMLBeans schema compilation options](#).

Setting Generated File Locations

The Generated File Locations tab provides a way for you to set the locations of JAVA and CLASS files generated from your WSDL or XSD files. (XMLBeans also generates other kinds of files, including XSB files, that it uses internally.) Typically, you'll want to leave these paths unchanged. (This user interface is also used when you add the facet to an existing project as well as in project properties for XMLBeans.)





The generated src file root box contains the name of the directory that will contain Java source files generated by XMLBeans. These generated source files are compiled into the standard Java output folder for your project (such as build/classes) along with the other compiled results of your project. The default is ".xbean_src".

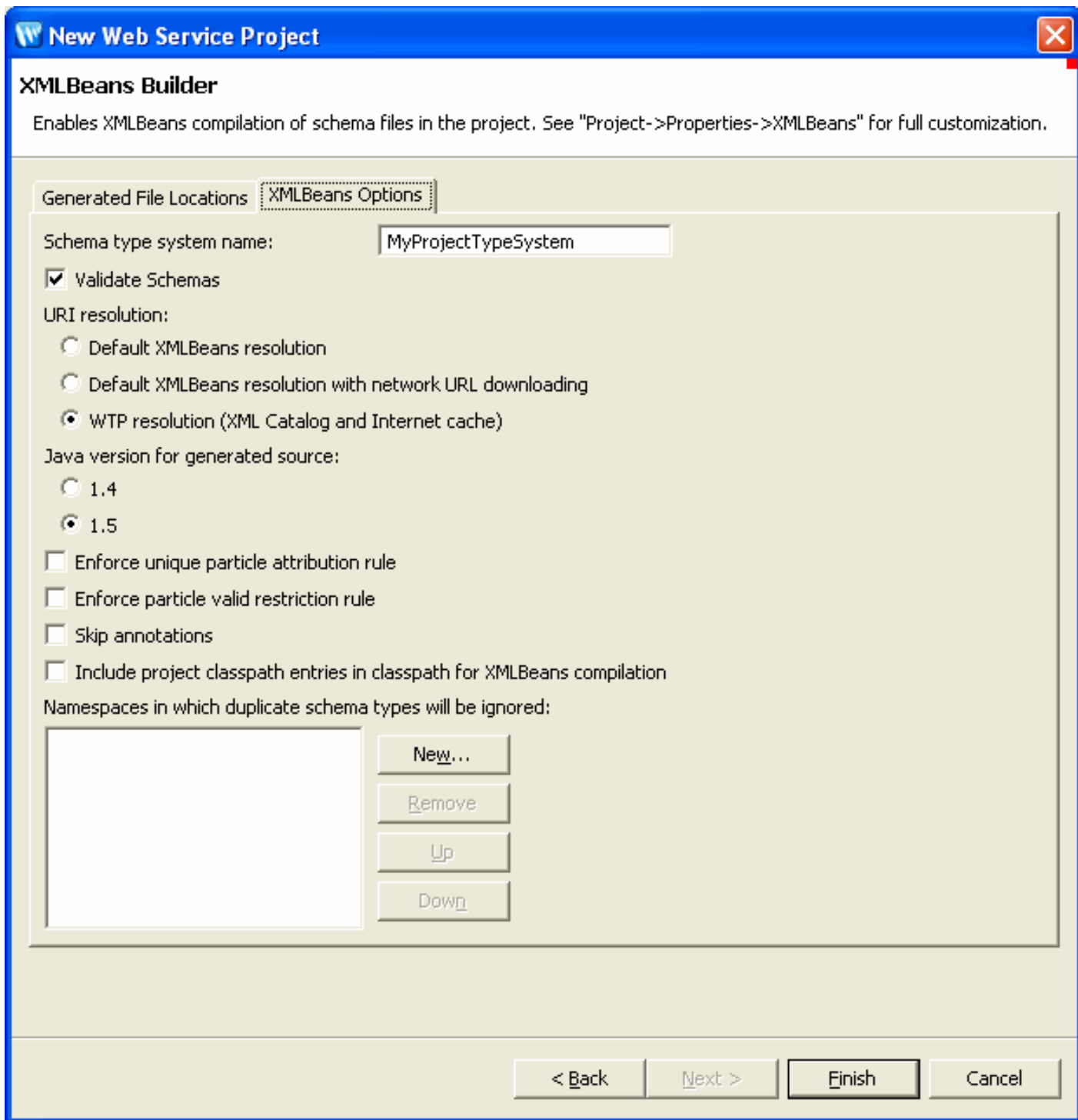
The generated binary file root box contains the name of the directory that will contain other files generated by XMLBeans. This directory is added to the project classpath and is exported to dependent projects. The default for the binary file root directory is ".xbean_bin".

Click the XMLBeans Options tab to set schema compilation options.

Setting XMLBeans Options

The XMLBeans Options tab provides a way for you to set options specific to schema compilation. Most of the options listed here are exposed by the open source XMLBeans schema compiler that the IDE uses, so you'll find them also documented in connection with the [xmlbean Ant task](#) at the Apache XMLBeans web site.

After you add the builder, you can change these and other settings through the project properties dialog. The XMLBeans category of project properties includes tabs for setting the generated file locations and XMLBeans options you set when adding the builder. (This user interface is also used when you add the facet to an existing project as well as in project properties for XMLBeans.)

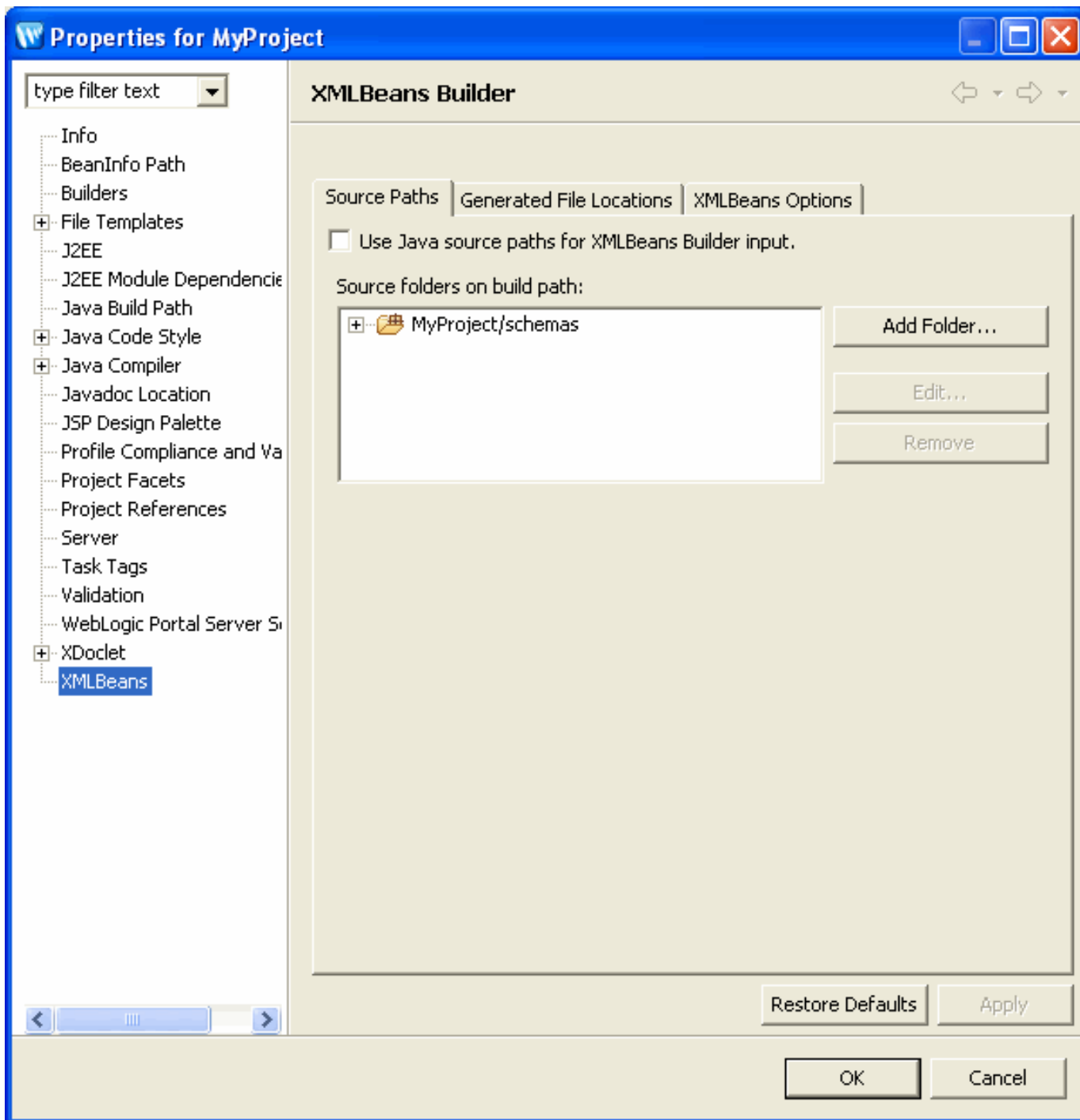


These options are described in the following table. (Note that the semantics for some of the Ant task attributes differs slightly from the options in the IDE user interface.)

XMLBeans Option	Description	Ant Task Attribute Counterpart
Schema type system name	Name of the schema type system (defaults to <projectname>TypeSystem)	typesystemname
Validate Schemas	Select this to have your schemas validated during compilation.	novdoc
URI resolution	Determines resolution of include/import declarations.	None.
Default XMLBeans resolution	An EntityResolver is not provided and network URL downloading is disabled.	Similar to "false" for the "download" attribute.
Default XMLBeans resolution with network URL downloading	Default XMLBeans resolution (in other words, no EntityResolver) but the network URL downloading flag is enabled.	Similar to "true" for the "download" attribute.
WTP resolution	EntityResolver is provided that delegates to the WTP URIResolver (searches the XML Catalog and Internet cache).	Similar to "false" for the "download" attribute.
Java version for generated source	Determines version compatibility of the generated Java source.	javasource
Enforce unique particle attribution rule	Determines if the unique particle attribution rule is enforced (in other words, no overlapping particles both in choice/all or validating adjacent items).	noupa
Enforce particle valid restriction rule	Determines if the particle valid restriction rule is enforced (in other words, whether one particle is an allowed restriction of another particle).	nopvr
Skip annotations	Determines if schema "<annotation>" elements should be skipped.	noann
Include project classpath entries in classpath for XMLBeans compilation	If true, compilation and xsdconfig processing will be performed with a classpath that includes the entries from the project classpath. This means that pre-compiled schema types on the classpath will be visible during compilation and Java classes on the classpath will be visible for interface extension processing.	No direct counterpart.
Namspaces in which duplicate schema types will be ignored	List of one or more namespaces in which duplicate definitions are to be ignored.	ignoreDuplicatesInNamespaces

Setting the Schema Compilation Source Path

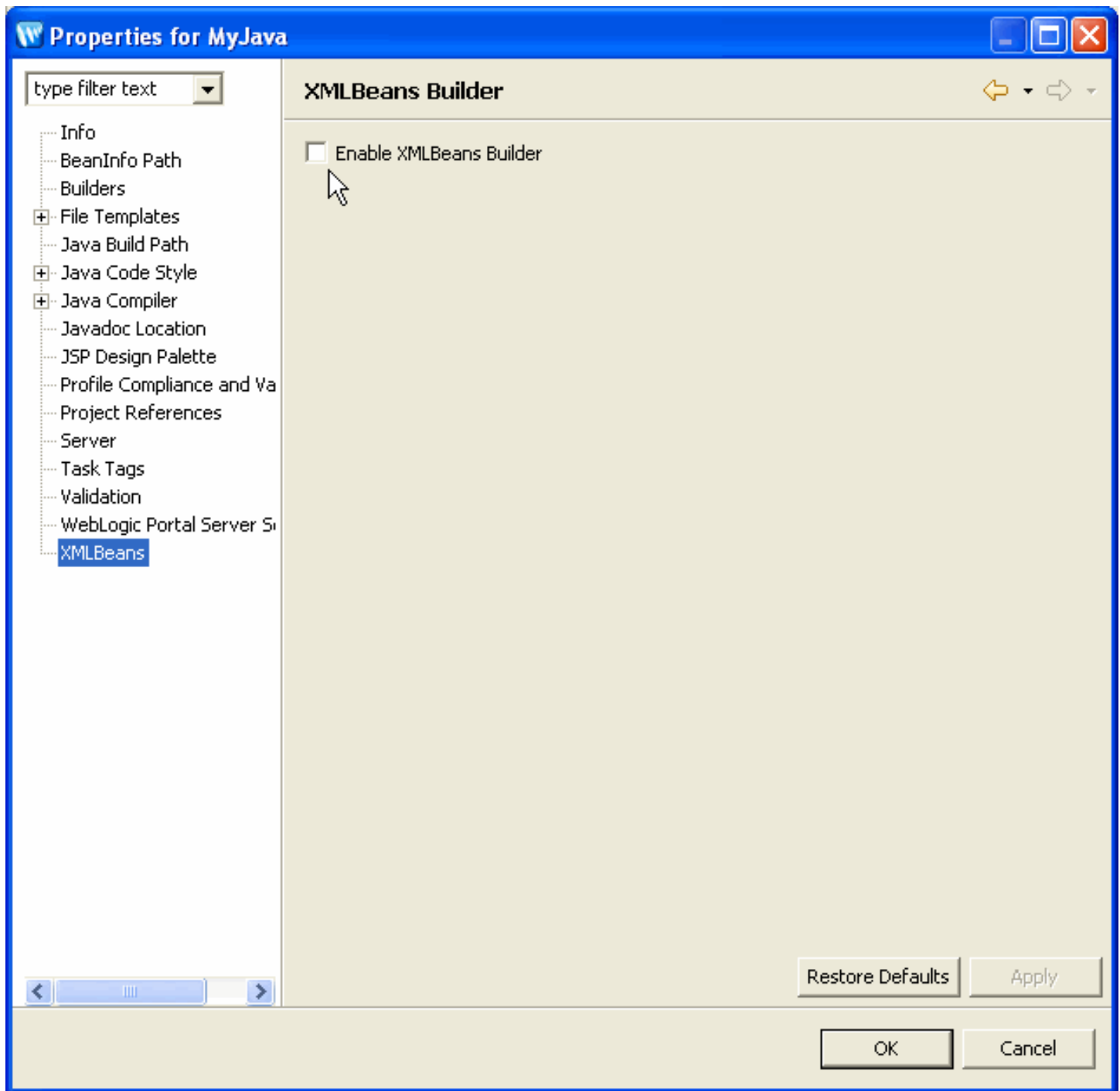
When compiling WSDL and XSD files into Java types, the IDE looks at directories on the schema source path for files to compile. This path is automatically set when you add the XMLBeans Builder facet, but you can make changes to it with the Source Paths tab, as shown in the following illustration. This is similar to a source path for JAVA file compilation. In fact, you can specify that those JAVA source paths themselves be used for locating WSDL and XSD files to compile.



Turning off Automatic Schema Compilation

If you don't want automatic schema compilation, you can disable the XMLBean Builder facet. This procedure for doing so differs between WTP faceted project and Java projects.

- Faceted projects — In the project's properties dialog, in the Project Facets page, clear the XMLBeans Builder check box that is shown in [Adding the XMLBeans Builder Facet](#).
- Java projects — In the project's properties dialog, in the XMLBeans page, clear the Enable XMLBeans Builder check box, as shown below.



Generating a JAR File that Contains Java Types Generated from Schema

If you want a JAR file that contains Java types generated from your schema, you can get it in one of two ways, as described below. Note that these produce slightly different results, so you should keep in mind what you'll be using the JAR file for.

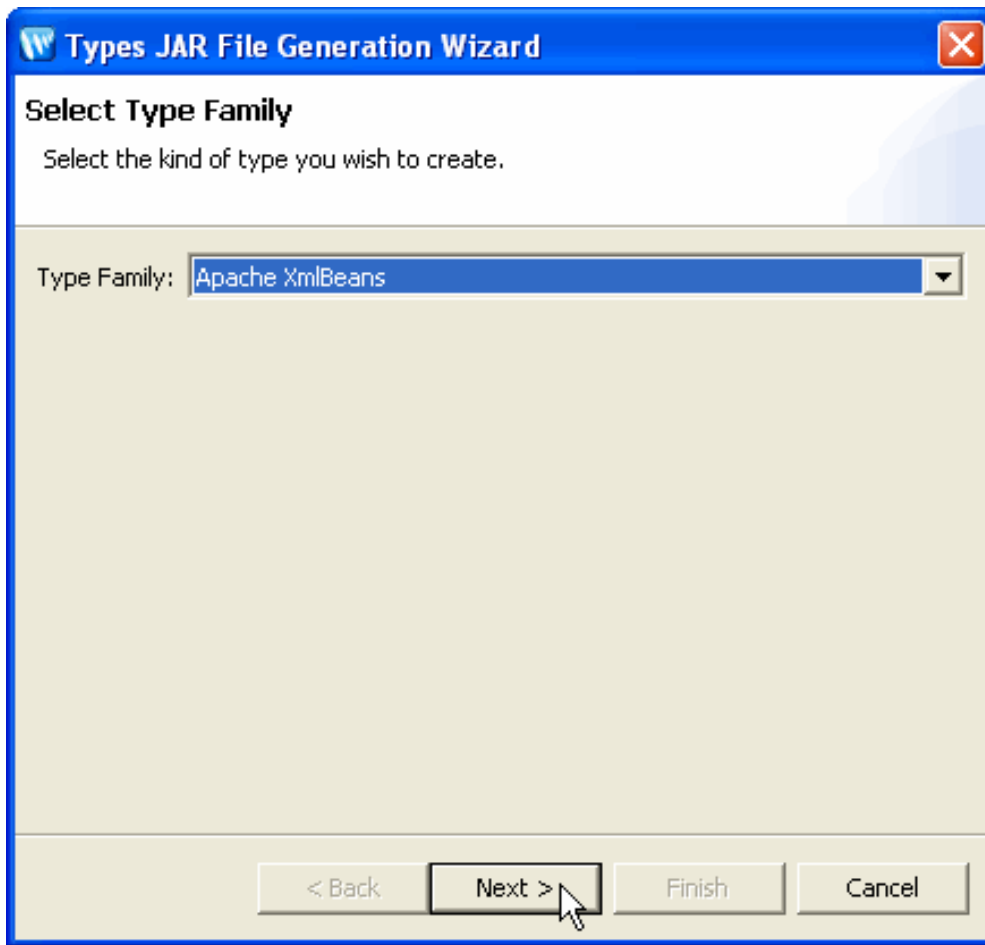
- Generate the Java types and put them into a JAR file in by creating a *types JAR*. This method compiles your schema and jars the resulting types. However, because it does not include artifacts such as an XSDCONFIG file, you might prefer the other method.
- Create a JAR file from types that were generated through automatic compilation. This method jars the existing XMLBeans Builder output, including XSDCONFIG files.

Creating a JAR File from Manually Generated Java Types

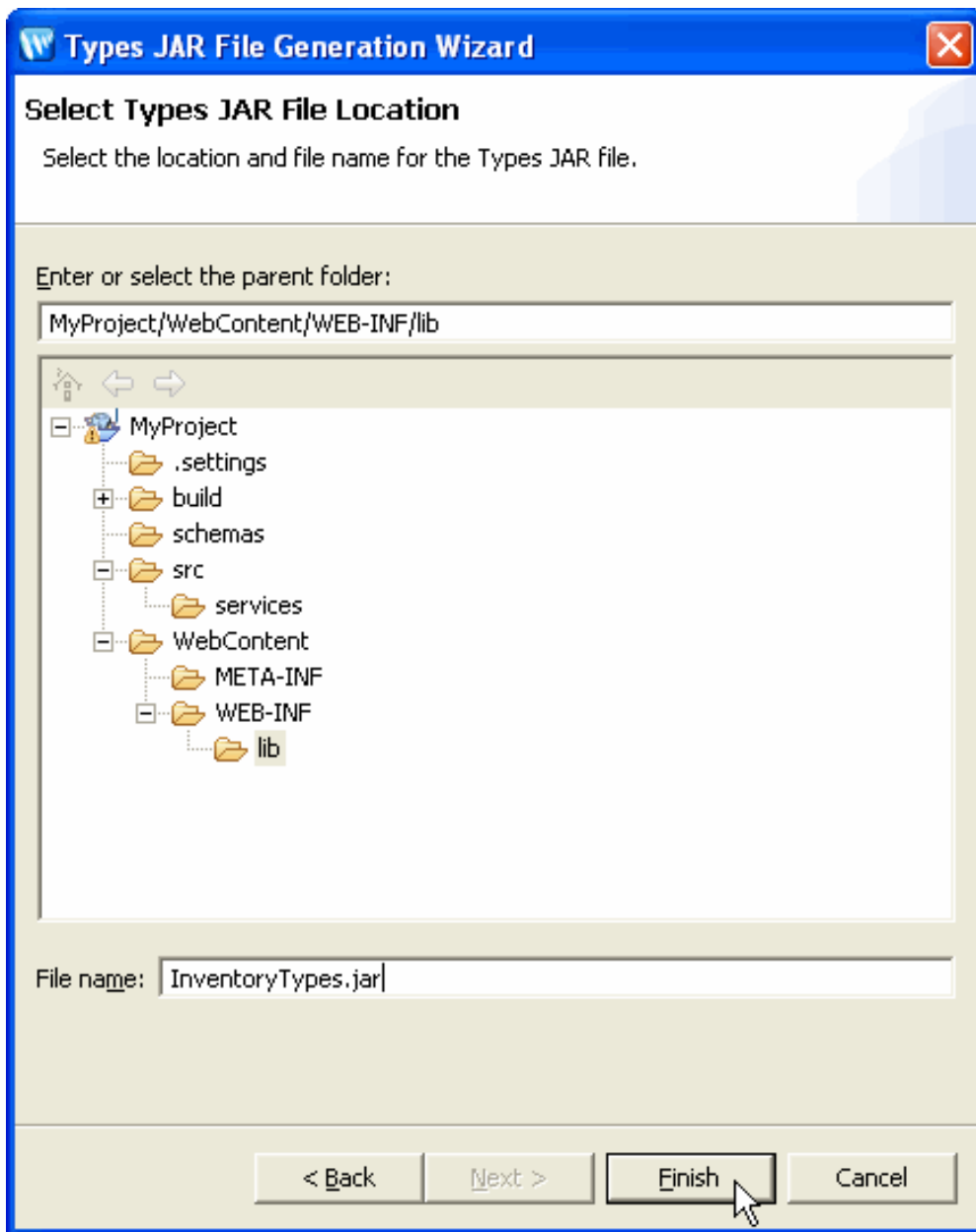
You can use an IDE menu command to generate a JAR file that contains the types when you want it. This is known as generating a *types JAR*. You might find this useful if you aren't using automatic compilation of Java types from schema. (Contrast this with the [procedure for getting a JAR file](#) from types that were generating through automatic schema compilation.)

Note: Be sure to see the [note above](#) for important information about types JARs.

1. In the Package Explorer, right-click the WSDL or XSD file whose generated Java types you want packaged into a JAR file, then click **Web Services > Generate Types JAR File**.
2. In the **Types JAR File Generation Wizard** dialog, in the **Type Family** box, ensure that **Apache XmlBeans** is selected, then click **Next**.



3. Select the parent folder for the JAR file you are creating, enter the JAR file's name in the **File name** box, then click **Finish**.



Creating a JAR from Automatically Generated Types

By using the following steps you can create a JAR file that includes generated Java types as well as the other XMLBeans-related artifacts you might need, including XSDCONFIG files. A JAR file created this way gets you the output of the XMLBeans Builder.

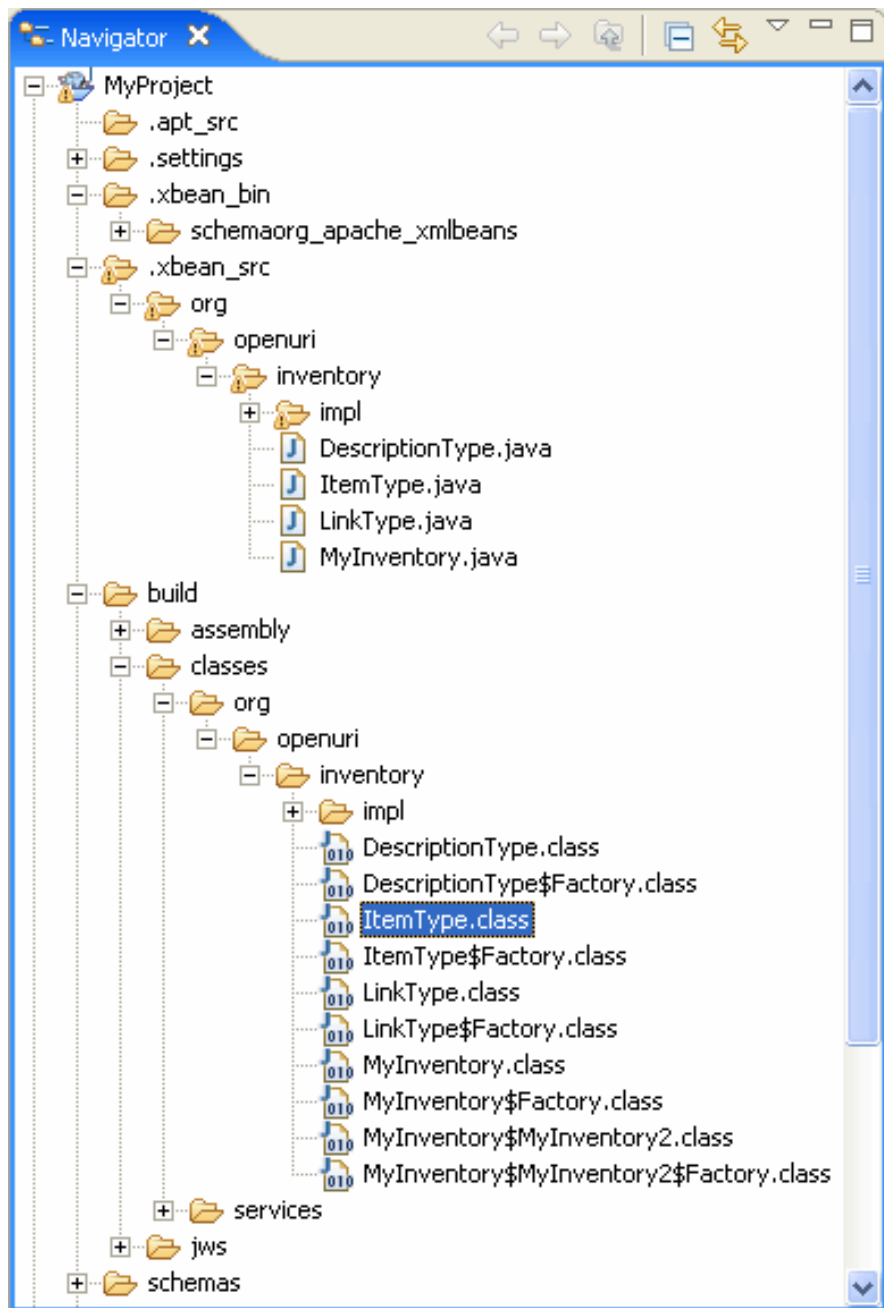
1. Right-click the project and select **Export**.
2. In the **Export** dialog, under **Select an export destination**, click **JAR file**, then click **Next**.
3. In the **JAR Export** dialog, under **Select the resources to export**, clear all but the **.xbean_bin** and **build** directories (make certain to deselect the various Eclipse metadata files under the project root). Note that these are the files that are used to build JARs for deployment.
4. Ensure that **Export generated class files and resources** is *not* selected.
5. Ensure that **Export all output folders for checked projects** and **Export .java source files and resources** are selected.
6. Click the **Browse** button to pick the name and location for your generated JAR.
7. Select **Add directory entries** (to be consistent with the JAR files created by the xmlbeans Ant task).

8. Click **Finish**.

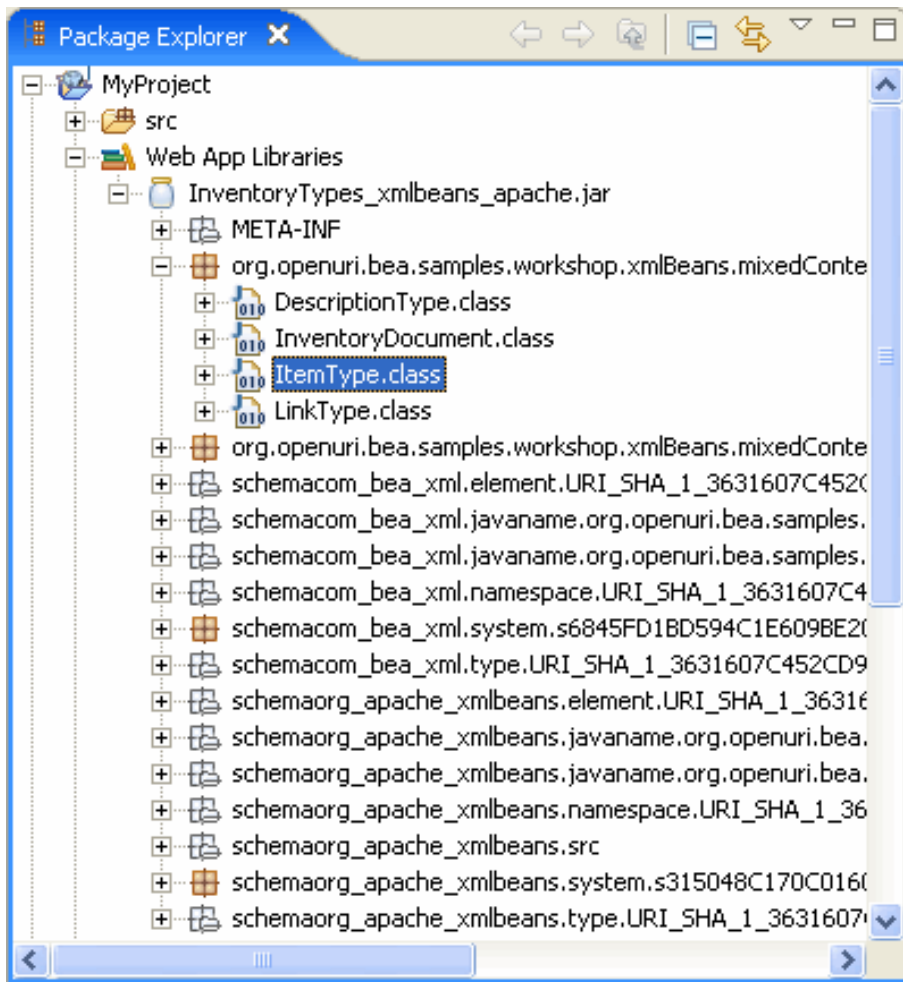
Locating Generated Java Types

Depending on how your project is set up to compile schema, the resulting Java types might not be easily discoverable in the IDE. For example, if you use the XMLBeans Builder facet to enable automatic compilation, the generated Java types are put into the generated file locations — which, by default, aren't visible in the Package Explorer or Project Explorer views.

XMLBeans Builder output — Use the IDE's Navigator view to locate Java source and binary files generated during schema compilation. The following illustration shows the default generated file paths as displayed in the Navigator view. Note, too, that compiled CLASS files are put into the build/classes directory with other classes resulting from your project's code.



Types JAR output — Use the Package Explorer view to browse the JAR's contents, just as you would with another JAR file. The following illustration shows how this might look.



Guiding XMLBeans Type and Package Naming During Compilation

When compiling a WSDL or schema into Java types, the XMLBeans builder will by default try to use your schema type names and URIs when naming generated Java types and Java packages, respectively. You can customize the names of resulting Java types and packages by using an XSDCONFIG file that maps a schema type name or namespace URI to a Java type name or package name that you specify. You put this file in the directory that holds the WSDL and XSD files you're compiling.

Coercing type naming in this way does not change the names for the underlying schema types — it merely determines the names for corresponding generated Java types. Selecting your own names for compiled types can be useful when you want to control type name length, locale, capitalization, and so on.

Note: Due to a Windows operating system limitation on file path lengths (256 characters), you might have trouble compiling schemas in which the generated types are nested in a very deep package hierarchy. This may result in an error message stating that a particular class couldn't be found. To work around this limitation, try using your XSDCONFIG file to guide the naming of generated packages so that the hierarchy is less deep, package names are shorter, and so on.

To Guide Type Naming During Compilation

1. Determine the names you want for each of the named types in your schema, including elements and attributes.

For example, you might decide that a PURCH_ORDER schema element and CUST schema type should be called PurchaseOrder and Customer as Java types.

2. Add a new XSDCONFIG file to your project.

1. Right-click the directory that contains the WSDL and XSD files that will be compiled to generated Java types, then click **New** > **File**.

This should be a directory that is on the [schema compilation source path](#) — by default, the "schemas" directory.

2. In the **New File** dialog, in the **File name** box, enter the name of your XSDCONFIG file, giving the file a name with an **XSDCONFIG** extension, then click **Finish**.
3. In the empty window that is displayed, enter the content of your XSDCONFIG file. For example, you might start with the following example, replacing the <namespace>, <package>, and <qname> elements with values that make sense for your needs.

```
<!-- An XSDCONFIG file must have a root "config" element in the
      http://xml.apache.org/xmlbeans/2004/02/xbean/config namespace. Also, be sure
      to declare any namespaces used to qualify types in your schema (here,
      the namespace corresponding to the pol prefix). -->
<xb:config xmlns:pol="http://openuri.org/easypoLocal"
          xmlns:xb="http://xml.apache.org/xmlbeans/2004/02/xbean/config">

  <!-- Use the "namespace" element to map a namespace to the Java package
        name that should be generated. -->
  <xb:namespace uri="http://openuri.org/easypoLocal">
    <xb:package>com.myco.potracker</xb:package>
  </xb:namespace>

  <!-- Use the "qname" element to map schema type names to generated
        Java type names. In these examples, the name attribute's value is the
        XML element name; the javaname attribute's value is the Java type
        that should be generated. -->
  <xb:qname name="pol:CUST" javaname="Customer"/>
  <xb:qname name="pol:PURCH_ORDER" javaname="PurchaseOrder"/>
</xb:config>
```

When you build the schema project, the compiler will use the XSDCONFIG file to define names for generated Java types.

Supporting the Interface Extension Mechanism

XMLBeans provides a mechanism through which interfaces generated during schema compilation will extend interfaces in your project. Likewise, generated implementation classes will implement extended interface methods via handler classes. You specify the interfaces to extend and the handler classes to use through an XSDCONFIG file. For more information on the interface extension mechanism, see the [XMLBeans wiki](#).

If you're using the interface extension mechanism, the IDE must of course know where your interfaces and handler classes are. You can take one of the following approaches:

- Put them on the [XMLBeans Builder's source path](#).
- Enable the XMLBeans option "[Include project classpath entries in classpath for XMLBeans compilation](#)" and ensure that the compiled classes for the handler and source files are visible on the project classpath.

Related Topics

None.

Tips and Tricks

Shortcuts and tips on how to use Workshop for WebLogic more effectively.

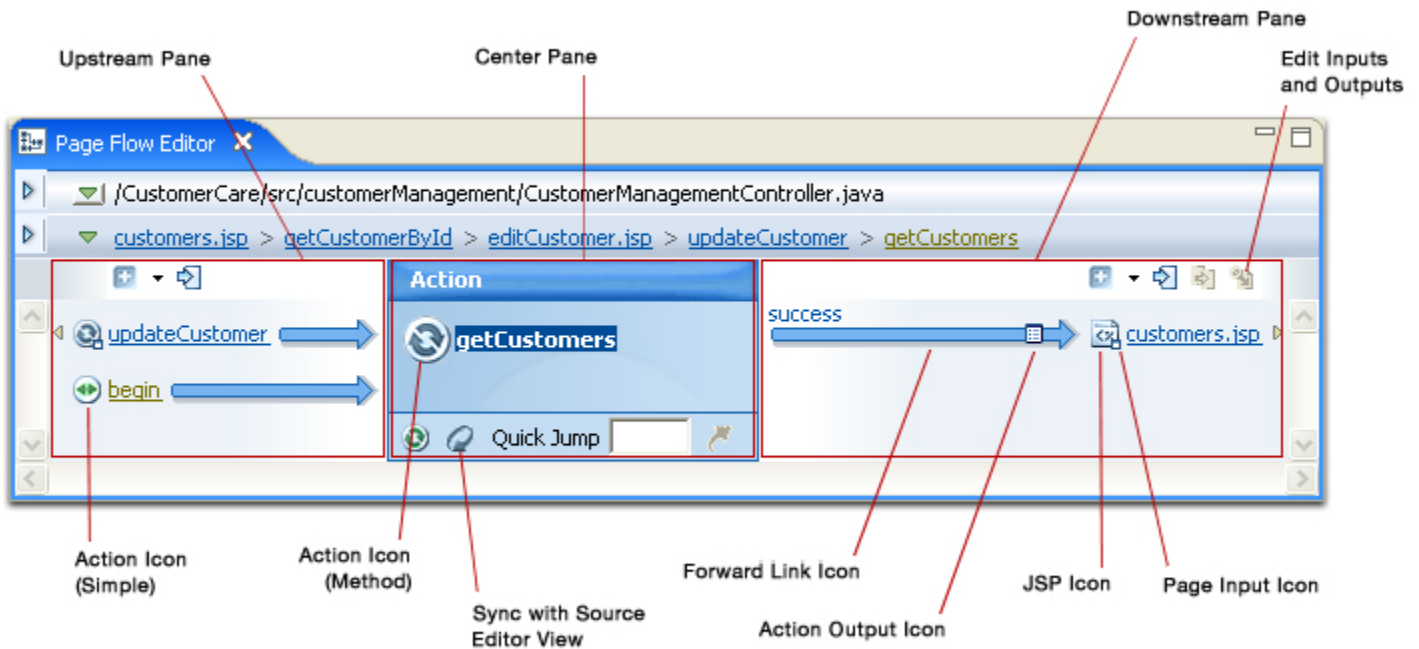
Links to sections:

[Page Flow Design](#)

[Streamlining Deployment/Testing](#)

[Controlling Builds](#)

Page Flow Design



What	How
Quick run	Click the Run on Server button in the toolbar of the Page Flow Explorer to run the current page flow, regardless of which source file is currently being edited.
Get agreement between action output annotations and corresponding page input tags	Use the Action Output/Page Input Editor to quickly get agreement between action output annotations and corresponding page input tags. Either select a forward link in the Page Flow Editor and click the Edit Inputs and Outputs button on toolbar over the downstream pane or right click on a forward link and select Edit Action Outputs
Create a hyperlink from the page that raises an action	Drag an action from the Page Flow Explorer and drop it downstream from a page in the center pane of the Page Flow Editor to create a hyperlink in the page that raises the action.
Create a forward from an Action to a JSP	Drag a page from the Page Flow Explorer and drop it downstream from an action in the center pane of the Page Flow Editor to create a forward from the action to the page
Change Page Flow Editor focus	Drag an action or page from the Page Flow Explorer onto the center pane of the Page Flow Editor to quickly change the focus of the editor.
Synch Page Flow Editor focus with current cursor position in the Source Editor	Use the Sync with Source Editor button at the bottom of the center pane of the Page Flow Editor to change the focus of the Page Flow Editor when the source editor cursor is in a node-related source element, such as an action method.

Setting the destination of the forward	Drag an action or page from the Page Flow Explorer onto an unspecified node to set the destination of the forward.
Replacing the destination of a page link	Drag an action from the Page Flow Explorer onto an existing action in the downstream pane of the Page Flow Editor to change the destination of the associated page link.
Replacing the destination of a forward	Drag a page node from the Page Flow Explorer onto an existing page in the downstream pane of the Page Flow Editor to change the destination of the associated forward.
Creating an action that calls a control method	Drag control methods from the Referenced Controls folder of the Page Flow Explorer onto the Page Flow Editor canvas to create an action that calls the control method.
Create a new node (JSP or Action)	Right click on the downstream pane of the Page Flow Editor to create a new node at that point in the flow diagram.
Edit a forward or page link	Right click on arrow links in the Page Flow Editor to see operations that apply to the forward or page link.
Display source elements by clicking corresponding icons	Double-click any icon in the Page Flow Editor or Page Flow Explorer to see the corresponding source element in the source editor
Edit form bean validation rules	Right click on any action that takes a form bean and select one of the validation rules options to launch the Validation Rule editor

Streamlining Deployment/Testing

What	How
Limiting the Projects that are Deployed	When you use the Run command, all <i>open</i> projects associated with the current server will be deployed. To streamline deployment, use the Project > Close Project command to close individual projects that are not needed for your current testing.
Using Working Sets to Limit the Projects that are Deployed	To specify a set of projects/files to be deployed, define a working set from the Navigator view.
Separating web services	<p>If you have a web service that calls another web service in the same project, the two web services should either be moved to separate projects (that are linked to separate EAR projects), or the shared code should be extracted into a control so that the web services do not call each other directly.</p> <p>A web services can call another web service in the same project, but this incurs a great deal of extra overhead that will slow down testing.</p>
Undeploy unneeded projects	From the Servers view, expand your server to see what projects are deployed to the server. On a Run command, all of these projects will be checked to verify that they are current. Right click on the server name and use the Add and Remove Projects command to remove unneeded projects.
Remove projects deployed to the server from other applications	<p>From the Servers view, double click on the server name. Then from the Servers Overview window in the Published Modules box, click on any projects that were loaded with other enterprise applications that are no longer needed and click Undeploy to remove them.</p> <p>Be sure that you do not remove any system library files. System files begin with "workshop", "beehive", "struts", etc. Remove only projects that you deployed.</p>

Controlling Builds

What	How
Disabling Automatic Build	If system performance is too slow, try Project > Build Automatically to disable automatic builds.
Cleaning up Build Artifacts	If builds get strange errors, try doing a Project > Clean before building to remove artifacts of previous builds

Troubleshooting

This topic describes techniques for avoiding problems and for resolving errors when they arise.

Top Ten Pitfalls to Avoid

1. Workshop for WebLogic is implemented in an open source environment, based on the Eclipse platform, and co-existing with software that comes from many other sources. As a result, there are many Eclipse commands and features that are **subsets** of the equivalent Workshop for WebLogic command or **don't work** with Workshop for WebLogic projects. If a command or feature does not work as expected, consult the Workshop for WebLogic documentation to determine the appropriate method to accomplish your task.
2. Testing and debugging server-based applications requires that you manage the files that are deployed to the server. Manage your server(s).
3. If your project does not build or deploy correctly, it may not have the correct dependencies on the enterprise application (EAR) project or it may not have the build path or the class path set correctly. Be sure that project dependencies are set correctly.
4. If you run two servers on the same domain, then effectively they are a single server and you may experience collisions and unpredictable behavior. Use separate domains if running two servers simultaneously.
5. Don't have two web services in a single project where one web service accesses the other-- this is very inefficient and slow.
6. Use the WebLogic project types (i.e., WebLogic EJB project rather than EJB project). When you create an EAR project, do not use the **New Modules** button because it creates Eclipse projects that do not support Workshop for WebLogic features.
7. Use **File > Import** to import archive files or projects. Use the upgrade wizard to import projects created with WebLogic Workshop 8.1.
8. Use **File > Export** to create archive files.
9. The Workshop for WebLogic features have been designed for use with Workshop for WebLogic features and display relevant information. Use the Workshop for WebLogic perspectives: Workshop, Page Flow. Other Eclipse perspectives that can be used are: Debug, Resource and J2EE.
10. You may deploy and test a dynamic web application as a standalone project or WAR. However all other project types (web services, EJB and utility projects) **MUST** be deployed through an Enterprise Application (EAR) project.

Potential Issues when Using Eclipse Commands with Workshop for WebLogic

When using Workshop for WebLogic, some standard Eclipse commands should not be used:

- Do not use **Project > Properties > Java Build Path** to set project dependencies. Java Build Path sets only Java build paths, which is a subset of the project dependencies that Workshop for WebLogic relies on. Click [here](#) for information on setting Workshop for WebLogic project dependencies.
- Do not use **File > Export > Ant Buildfiles**. Use **File > Export > Workshop Ant Script** and **File > Export Workspace Metadata for Workshop Ant Script**.
- When creating an EAR project, the final screen shows a **Create Modules** button that creates projects that are not enabled for Workshop for WebLogic features. Do not use the **Create Modules** button.
- Do not use **File > New > Project > Web > Static Web Project** for building Workshop for WebLogic web applications. Use **File > New > Project > Web > Dynamic Web Project** .
- When running a Workshop for WebLogic application, use **Run As > Run on Server**.

General Procedures for Troubleshooting

When troubleshooting, follow these steps to diagnose your problem:

1. Check at <http://edocs.bea.com/workshop/docs92/relnotes/index.html> to see the known issues for this version of Workshop for WebLogic.
2. Review the release notes for Beehive, WTP and Eclipse.
3. Review the sample code and tutorials supplied to get ideas for better practices. Be sure to start the help in a standalone window that is not affected by restarts.
4. Check the **Problems** view to verify that the application is being built correctly by Workshop for WebLogic. Use the **Quick Fix** feature to resolve errors.
5. Check the documentation to be sure that you are deploying your application correctly.
6. Use test client to test web services directly rather than through another application.

Problems with Project Structure (Build Errors)

If your projects compile correctly in the IDE but generate a **Class Not Found** exception when running on the server, you need to set module dependencies by right clicking on project name in the **Package Explorer** view and choosing **Properties**. Click **J2EE Module Dependencies** and set EJB/utility project links. See [Managing Project Dependencies](#) for more information.

Be sure to link your projects to an EAR project correctly. See [Managing Project Dependencies](#) for more information.

The **Project > Clean** command removes old build artifacts that may cause the current build to fail.

Problems with Deploying

If you are running two servers simultaneously, the servers use the same port, resulting in collisions and contention. To use two servers simultaneously, be sure to define each server on a different domain.

If your project(s) won't deploy with the error: "Module xx failed to deploy!" you may have a duplicate project name already loaded on the server. This may happen if you go through a tutorial more than once, or if you have standardized modules that you use in more than one application. See Managing Servers for more information.

If deployment is slow, you may want to use working sets to manage what is deployed. If you close a project in the current workspace (**Project > Close Project**) , it will not be deployed. You may also want to undeploy previous projects.

Usage Reporting FAQ

The IDE includes a feature through which it can, with your permission, collect data about IDE usage. This topic answers questions related to this feature, describing the information collected.

Note: Only information permissible under BEA's [privacy policy](http://www.bea.com) (at www.bea.com) will be collected by the usage reporting feature.

Questions

1. [What is the usage reporting feature?](#)
2. [What kind of information will be gathered?](#)
3. [What will BEA do with this information?](#)
4. [Will any personal information will be gathered?](#)
5. [Can I stop participating later if I choose?](#)
6. [Can I see the information that is being gathered?](#)
7. [How is the usage reporting feature turned on or off?](#)

Answers

1. **What is the usage reporting feature?**

This is a feature through which the IDE gathers information about certain user actions and sends the information to BEA. The IDE team uses the information to improve future versions of the IDE.

2. **What kind of information will be gathered?**

The information gathered includes:

- The version of Workshop being used.
- JVM name and version, and classpath.
- Operating system name and version.
- Number of processors and amount of installed memory.
- The Workshop and Eclipse plug-ins installed and active.
- The heapsize dedicated to Workshop.
- Number of projects open within Workshop, and the natures and facets defined for those projects.

- o The server(s) being targeted by Workshop projects.

3. **What will BEA do with this information?**

BEA Systems will use the information gathered to help us focus Workshop development on those areas of greatest value to our customers.

4. **Will any personal information will be gathered?**

No. The information gathered cannot be traced back to a particular system or user.

5. **Can I stop participating later if I choose to?**

Yes. At any time, you can choose the **Help > Usage Reporting** menu command, and opt out using the resulting dialog. No further information will be gathered unless and until you opt back in.

6. **Can I see the information that is being gathered?**

Yes. The last block of data reported to BEA Systems' usage reporting server is stored in:

```
<WORKSHOP_HOME>/workshop4WP/eclipse/configuration/com.bea.wlw.usagetrack.startup/  
last_message.xml
```

The block will be overwritten with new data each time a report is sent.

7. **How is the usage reporting feature turned on or off?**

The feature is turned on by default. During initial launch, the user interface provides an option to turn reporting off; to do so, be sure to make the appropriate change in the user interface and click OK.

After you install the IDE, you can turn the feature on or off through the IDE preferences dialog box.

1. Click **Help -> Usage Reporting**.
2. In the **Usage Reporting** dialog box...

Related Topics

[BEA Privacy Policy](#)