# Web Development Using Borland® JBuilder® 8 and IBM® WebSphere® Application Server Advanced Edition 4.0

Jump-start development, deployment, and debugging
Servlets,™ JSP,™ and Struts

**A Borland White Paper**

*By Sudhansu Pati, Systems Engineer, Borland Software Corporation*

December 2002

Borland®

## Contents

## Introduction

Borland® JBuilder® Enterprise is a comprehensive set of award-winning visual development tools for creating enterprise-scale applications written entirely in the Java™ programming language for the Java 2 platform.

JBuilder 8 Enterprise provides out-of-the-box integration with the market-leading J2EE™ platform application servers like Borland® Enterprise Server 5.x, BEA WebLogic Server™ 5.1, 6.x, 7.0, IBM® WebSphere® 3.5, 4.0, iPlanet™ 6.x, Sybase® Enterprise Application Server 4.1. Also JBuilder 8 has a plug-in to integrate with the Oracle 9i™ Application Server 9.0.3.

JBuilder 8 Enterprise ships with Tomcat 3.x and 4.x, and provides end-to-end integration with Tomcat.

This paper provides an overview of creating, testing, and debugging Web applications using Borland JBuilder 8 Enterprise and IBM WebSphere Application Server Advanced Edition 4.0.

The other platforms and software used to create the exercises are Optimizeit™ Suite 5.0, Windows XP,™ and Microsoft® Internet Explorer 6.0.

Refer to the white paper "Borland JBuilder 8 Configuration with J2EE Application Servers" for installation and configuration instructions for integrating JBuilder 8 Enterprise with WebSphere Advanced Edition 4.0.

WebSphere runs on the IBM JDK® (**Java Version J2RE 1.3.0 IBM Build cn130-20010601**) and JBuilder is capable of creating, deploying, testing, and optimizing Web applications hosted on the IBM JDK.

Followings are the list of Web technologies that can be easily developed, debugged, and tested using JBuilder 8 wizards. Web applications developed with any other technologies not mentioned here can also be easily imported to JBuilder 8 and managed.

| Technology | Description |
|---|---|
| **Servlets** | A server-side Java application which can process requests from clients. |
| **JavaServer Pages™ (JSP™)** | An extension of servlet technology. JavaServer Pages use custom tag libraries and offer a simplified way to develop servlets. |
| **InternetBeans Express** | A set of components and a tag library provided by Borland, used for easier presentation and manipulation of data from a database. |
| **Struts** | An open source tag library provided by the Jakarta Project that is used for building Web applications. Struts provides a flexible control layer based on standard technologies like servlets, JSP, JavaBeans,® ResourceBundles, and XML. |
| **JavaServer Pages Standard Tag Library (JSTL)** | A tag library provided by Sun that is part of the Java Web Services Development Pack 1.0 (WSDP). It provides a set of tags that allow developers to do common tasks in a standard way. The JSTL consists of four areas, each with its own TLD (tag library descriptor) and namespace. |
| **Cocoon** | A servlet-based, Java publishing framework for XML that is integrated into JBuilder. Cocoon allows separation of content, style, and logic and uses XSL transformation to merge them. It can also use logic sheets, Extensible Server Pages (XSP), to deliver dynamic content embedded with program logic written in Java. |
| **Applets** | A specialized kind of Java application that can be downloaded by a client browser and run on the client's machine. |

***Table 1****: List of major Web technologies supported by JBuilder 8*

## Working with JSP™

### Create a project

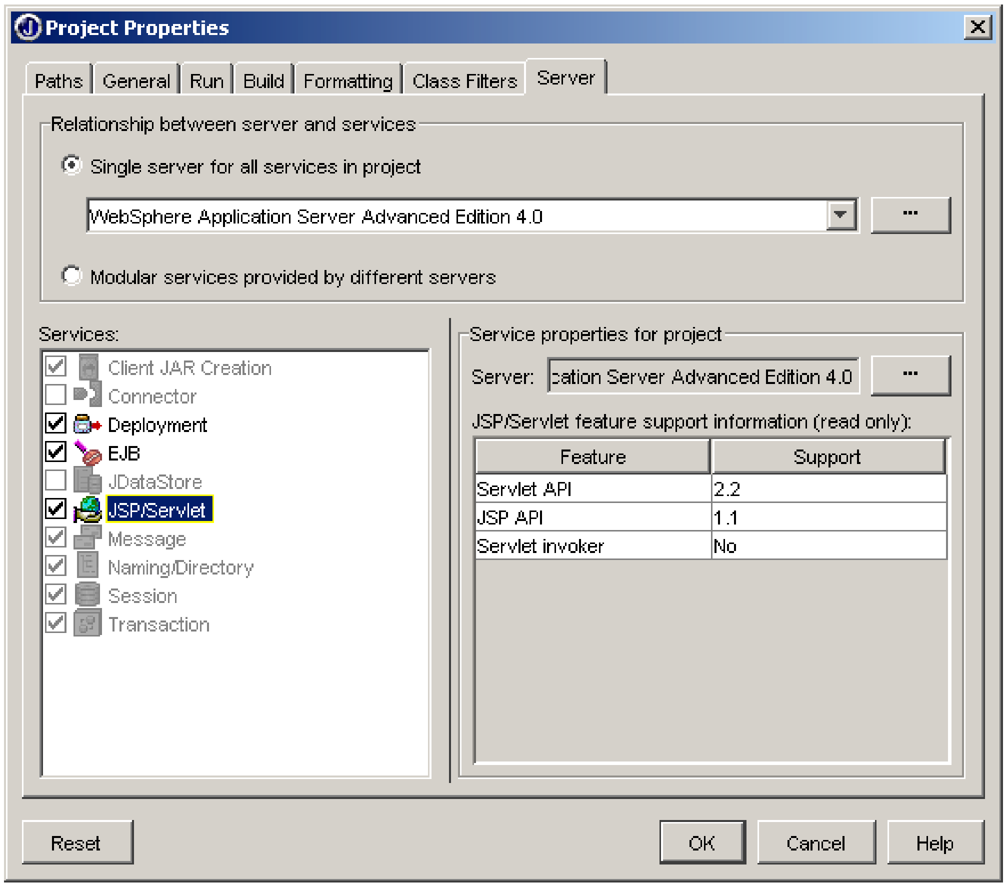
Start JBuilder 8. Click **File** -> **New** -> Select **Project** tab from the Object Gallery. Click **Project**. Click **OK**. In the **Project Wizard—Step 1 of 3** specify a Directory Name and a Project Name. Leave the Template as (Default Project). Click **Finish**.

For this exercise, the project directory is **`E:\DEMO\WebSphere\TestWebProjet`** and Project Name is **TestWebProject**.

## Attach WebSphere® to current project

You can attach the WebSphere Application Server to the current project. From the JBuilder main menu bar, click **Project** -> **Project Properties**. Click **Server** tab. Select **WebSphere Application Server Advanced Edition 4.0** from the combo box.

See Figure 1 for details.



Relationship between server and services

(●) Single server for all services in project

WebSphere Application Server Advanced Edition 4.0

( ) Modular services provided by different servers

Services:

- ✓ Client JAR Creation
- ☐ Connector
- ✓ Deployment
- ✓ EJB
- ☐ JDataStore
- ✓ JSP/Servlet
- ✓ Message
- ✓ Naming/Directory
- ✓ Session
- ✓ Transaction

Service properties for project

Server: cation Server Advanced Edition 4.0

JSP/Servlet feature support information (read only):

| Feature | Support |
|---|---|
| Servlet API | 2.2 |
| JSP API | 1.1 |
| Servlet invoker | No |

***Figure 1**: Project Properties window allowing users to select application server of choice*

**Note**: The **Services** panel in the left-hand side show the services provided by the specific server selected in the combo box (this case WebSphere Application Server AE 4.0). JBuilder is intuitive and automatically discovers the services provided by a selected Web or application server. In this case, deployment, EJB,™ and JSP/servlet are enabled because these services are available with WebSphere AE 4.0.
**Note**: You can click on each service from the **Service** panel and see what specific versions of different technologies are supported by the chosen Web or application server. In the above figure, the **JSP/servlet** service is clicked from the **Service** panel and JBuilder is showing the supported JSP and servlet versions with WebSphere AE 4.0.

**Note**: JBuilder provides users the option of choosing one server for all services in the project or multiple servers providing different servers. With **Modular Services provided by different servers**, users can configure different services to be run by different servers. For example: users can select Tomcat for JSP/servlet server, Borland® Enterprise Server for EJB and another server for connector.

For our exercise, leave the default, which is **Single server for all services in the project**.

Click on **Deployment** from the **Services** panel. Note that by default, the **Build target** for deploying to server is **Make**. If you prefer not to build the archives at the time of deployment, choose <**None**> from the combo box. This will stop building the project during deployment and make the deployment faster.

Click **OK** and JBuilder project **TestWebProject** is now configured for WebSphere Application Server AE 4.0.

## Attach the WebSphere® Application Server to all the projects

If you decide to use WebSphere AE 4.0 for all of your projects, you can set this as default. From JBuilder menubar click **Project** -> **Default Project Properties**. Click **Server** tab. Select **WebSphere Application Server Advanced Edition 4.0** from the combo box and choose **Single server for all services in the project**.

## Create a Web application

JBuilder 8 allows users to create a Web application that can host applets, Web Start launchers, JSPs, servlets, and Apache™ Struts applications.

From the JBuilder main menubar, click **File** -> **New** -> **Web** tab in **Object Gallery** -> **Web Application**. Click **OK**..

See Figure 2 for JBuilder 8 Object Gallery.

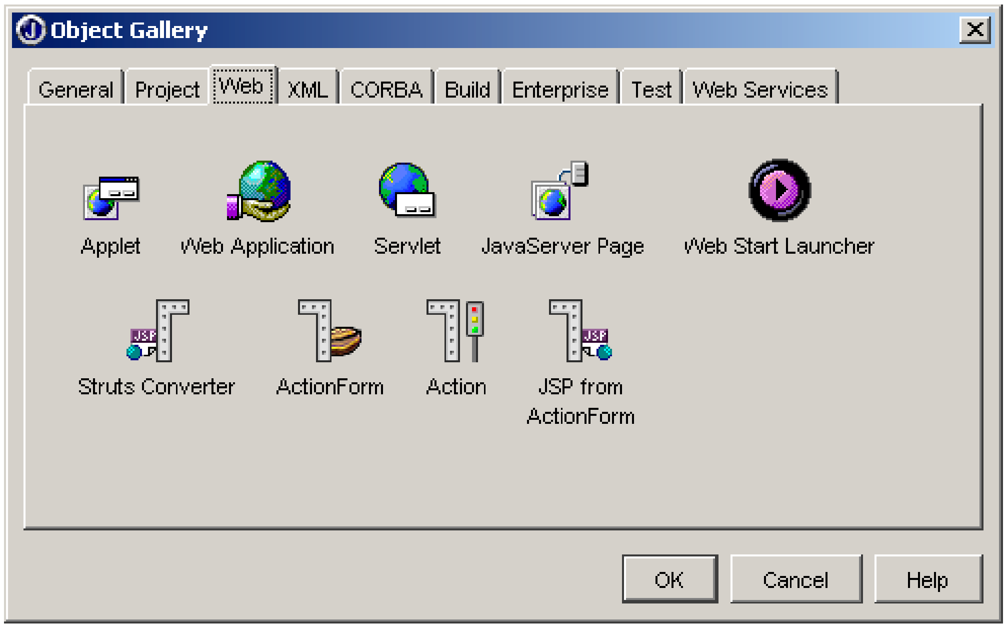


***Figure 2****: JBuilder 8 Object Gallery displaying the wizards to create Web applications*

The **Web Application Wizard** pops up. Enter **Name** and **Directory** entries for the Web application. For this exercise, the Web application name is **TestWebApp** and directory is **TestWebApp**.

JBuilder provides an option to choose when the developers would like to build a WAR file. For this exercise, leave the default which is **When building project or Web app**.

From the JBuilder 8 project tree, expand the project, and expand the Web application **TestWebApp**. Expand the **deployment descriptors** folder. Note that JBuilder has recognized that the target server is WebSphere, so it automatically created the following deployment descriptors for WebSphere.

```
ibm-web-bnd.xmi
ibm-web-ext.xmi
web.xmi
```

Click on **web.xml**. This opens the **WebApp DD editor** in the editor pane.

See Figure 3 for details.

**Note**: The Deployment Descriptor Editor tool in JBuilder provides a graphical user-friendly way to update the web.xml, and it is a two-way tool. It automatically changes the XML source when the Deployment Descriptor Editor is changed.

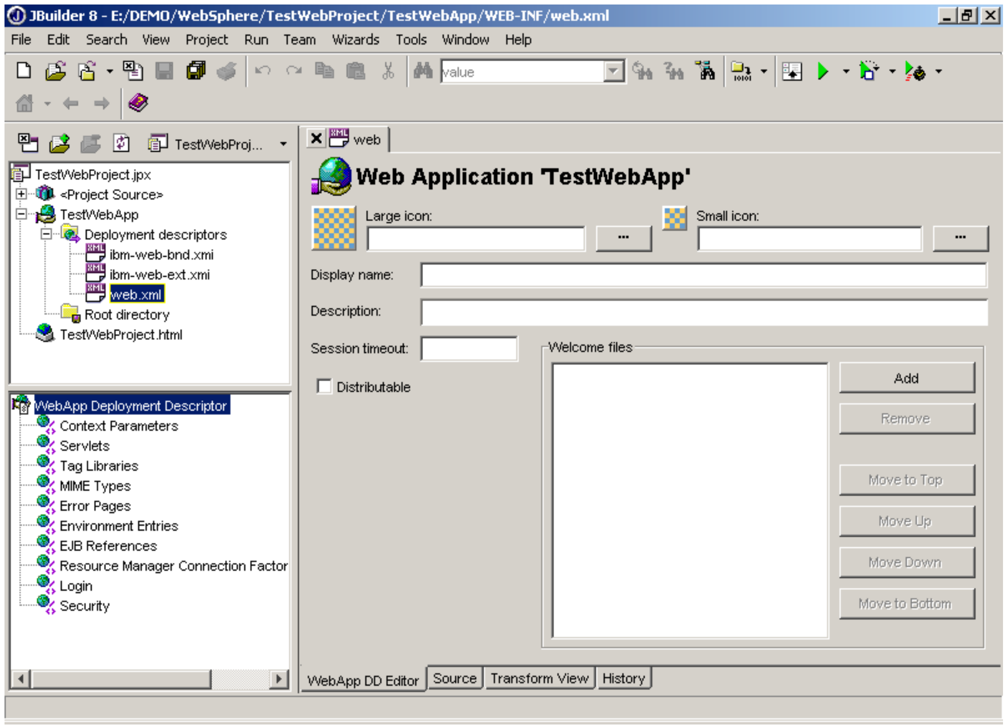


***Figure 3**: JBuilder 8 Web Application Deployment DescriptorEditor*

## Create a JSP™

From the JBuilder main menubar, click **File** -> **New** -> **Web** tab in **Object Gallery** -> **JavaServer Page**. Click **OK**..

In **JSP Wizard—Step 1 of 6**, enter a JSP name. For this exercise, the JSP name is **TestJSP**. Note that the Web application **TestWebApp** is defaulted for hosting this JSP. You can check both the checkboxes to generate a **sample bean** and an **error page**. Click **Next**.

In **JSP Wizard—Step 2 of 6**, you can select **InternetBbeans technology**, different tag libraries from **JSTL 1.0** and **Struts**. Leave the defaults and click **Next**.

In **JSP Wizard—Step 3 of 6**, enter the package name as **com.borland.demo.web**.
Click **Next**.

At this point, you can click Finish. JBuilder automatically creates a runtime configuration named **TestJSP** to run the created JSP.

**Note**: A runtime configuration is a configuration to run a runnable application in JBuilder 8. Runtime configuration is essential when your project has multiple runnable files or applications, and you want to run a specific application.

## Create an EAR

WebSphere requires Web or EJB applications to be deployed as EAR. JBuilder has a wizard to create EAR files easily.

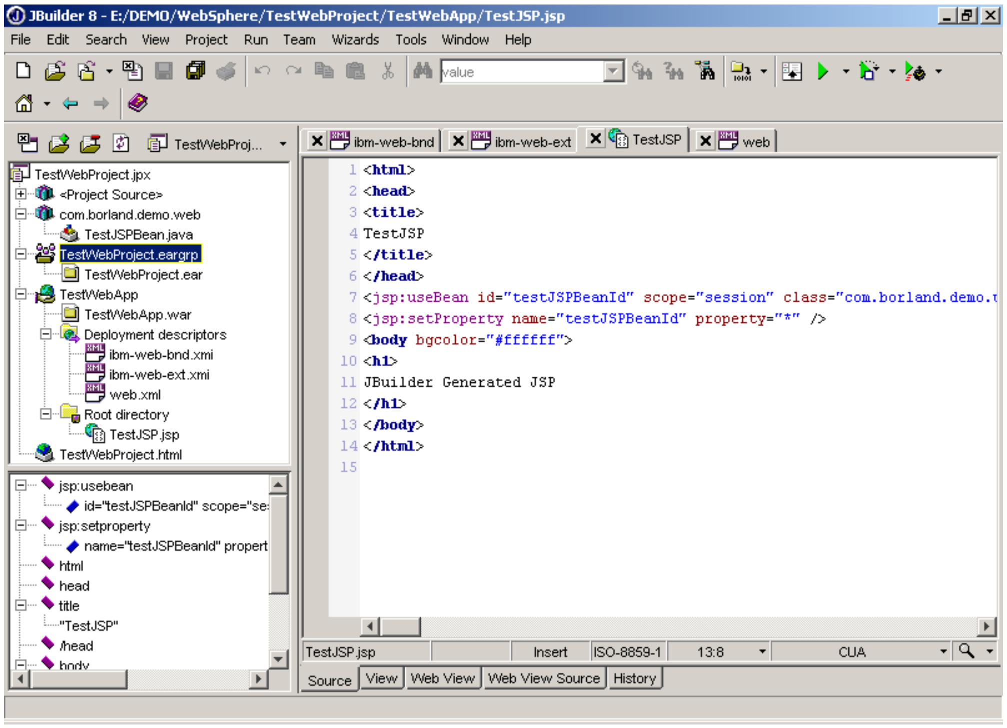
Click **File** -> **New** -> **Enterprise** tab of **Object Gallery** -> click **EAR** -> click **OK**.

In the **EAR Wizard**, keep clicking **Next** until you get **EAR Wizard–Step 5 of 6**. Check the checkbox under **Include** column. This includes the Web application **TestWebApp** in the EAR file. Click **Finish**.

## Make the project

To make the project, press **Ctrl+Shift+F9** or click the **Make** icon from the JBuilder toolbar. As a result of make, JBuilder automatically creates the WAR file **TestWebApp.war** and EAR file **TestWebProject.ear** for the Web application.

See Figure 4 for details.



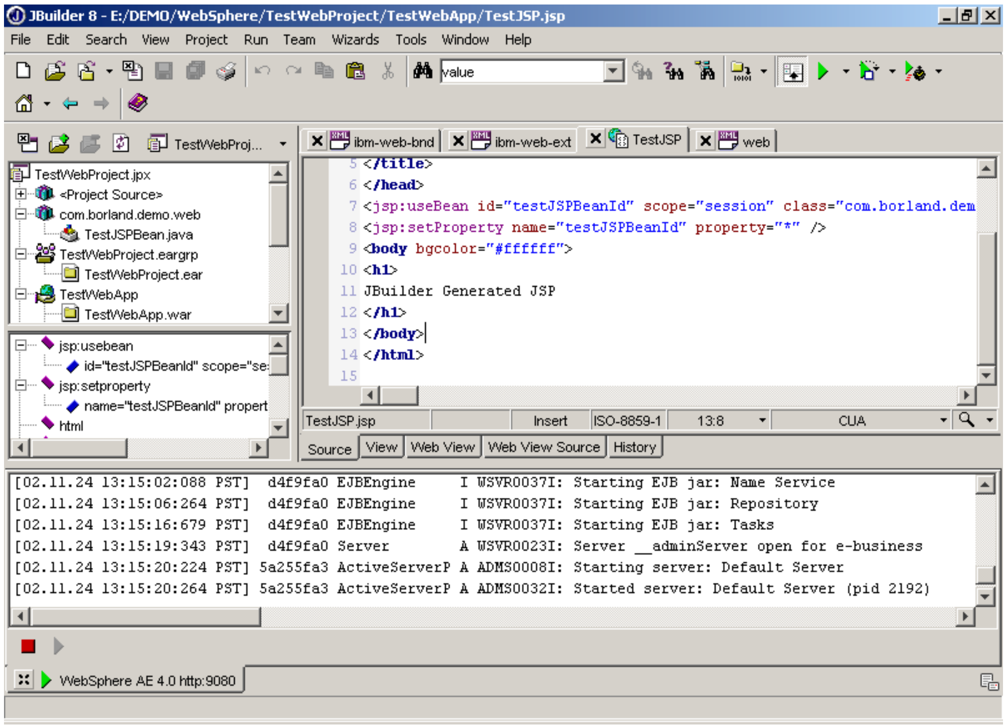
```
<html>
<head>
<title>
TestJSP
</title>
</head>
<jsp:useBean id="testJSPBeanId" scope="session" class="com.borland.demo.w
<jsp:setProperty name="testJSPBeanId" property="*" />
<body bgcolor="#ffffff">
<h1>
JBuilder Generated JSP
</h1>
</body>
</html>
```

*Figure 4*: *JBuilder showing the generated sources, WAR file, and deployment descriptors for WebSphere*

## Deploy and run the JSP™ in WebSphere®

Start WebSphere within JBuilder. To start WebSphere within JBuilder, click on the **Run Project** icon from JBuilder toolbar and click **TestJSP**.

See Figure 5 for details.



```
</title>
</head>
<jsp:useBean id="testJSPBeanId" scope="session" class="com.borland.dem
<jsp:setProperty name="testJSPBeanId" property="*" />
<body bgcolor="#ffffff">
<h1>
JBuilder Generated JSP
</h1>
</body>
</html>
```

```
[02.11.24 13:15:02:088 PST]  d4f9fa0 EJBEngine     I WSVR0037I: Starting EJB jar: Name Service
[02.11.24 13:15:06:264 PST]  d4f9fa0 EJBEngine     I WSVR0037I: Starting EJB jar: Repository
[02.11.24 13:15:16:679 PST]  d4f9fa0 EJBEngine     I WSVR0037I: Starting EJB jar: Tasks
[02.11.24 13:15:19:343 PST]  d4f9fa0 Server        A WSVR0023I: Server __adminServer open for e-business
[02.11.24 13:15:20:224 PST] 5a255fa3 ActiveServerP A ADMS0008I: Starting server: Default Server
[02.11.24 13:15:20:264 PST] 5a255fa3 ActiveServerP A ADMS0032I: Started server: Default Server (pid 2192)
```

***Figure 5***: *WebSphere AE 4.0 started within JBuilder*

Now the EAR file can be deployed to WebSphere Application Server AE 4.0. To deploy the EAR file, right-click **TestWebProject.eargrp** -> click **Deploy Options for "TestWebProject.ear"** -> **Deploy**.

See Figure 6 for details.

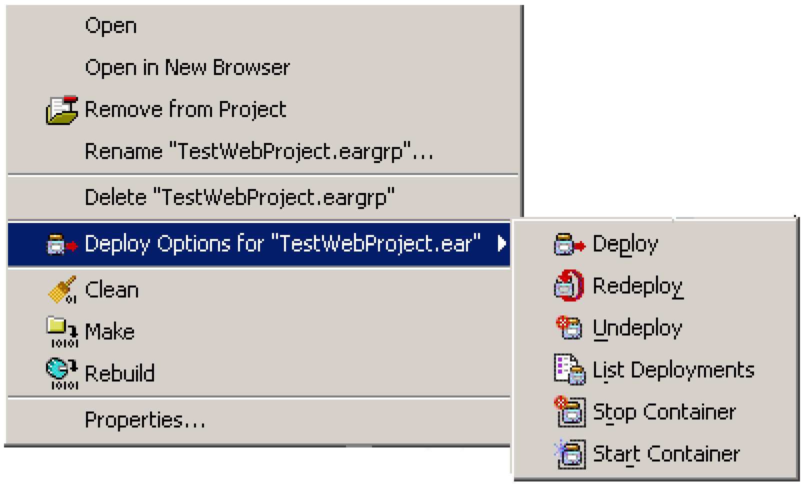


***Figure 6**: Deployment options to deploy archives to WebSphere*

Another tab called **WebSphere Application Server Advanced Edition 4.0 Enterprise Deployer** opens up in JBuilder message pane.

See Figure 7 for details.

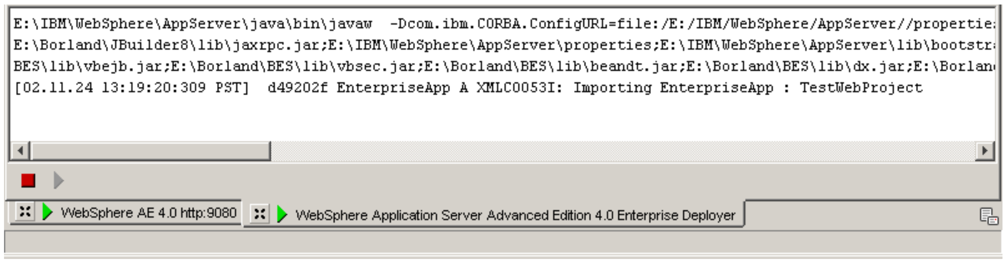

```
E:\IBM\WebSphere\AppServer\java\bin\javaw  -Dcom.ibm.CORBA.ConfigURL=file:/E:/IBM/WebSphere/AppServer//propertie
E:\Borland\JBuilder8\lib\jaxrpc.jar;E:\IBM\WebSphere\AppServer\properties;E:\IBM\WebSphere\AppServer\lib\bootstr
BES\lib\vbejb.jar;E:\Borland\BES\lib\vbsec.jar;E:\Borland\BES\lib\beandt.jar;E:\Borland\BES\lib\dx.jar;E:\Borlan
[02.11.24 13:19:20:309 PST]  d49202f EnterpriseApp A XMLC0053I: Importing EnterpriseApp : TestWebProject
```
WebSphere AE 4.0 http:9080
WebSphere Application Server Advanced Edition 4.0 Enterprise Deployer

***Figure** 7: WebSphere Application Server Advanced Edition 4.0 Enterprise Deployer tab showing deployment status to WebSphere*

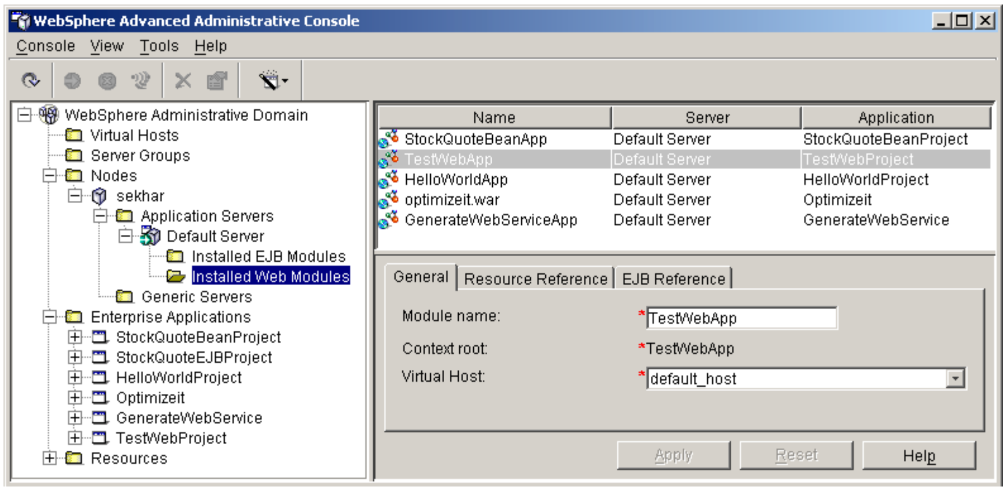
To run the JSP, right-click **TestJSP.jsp** from JBuilder project tree, click **Web run using TestJSP.jsp**

The AppBrowser runs the JSP and displays the message **JBuilder Generated JSP**.

If you like to see the deployment status in WebSphere console, start the WebSphere console by clicking **Tools** -> **WebSphere 4.0 AE Administrator's Console**.

Expand **Nodes** -> <**ServerName**> -> **Application Servers** -> **Installed Web Modules,** and you can find the Web application **TestWebApp** in the right-hand window. Also the console displays the status of the Web application.

See Figure 8 for details.



| Name | Server | Application |
|---|---|---|
| StockQuoteBeanApp | Default Server | StockQuoteBeanProject |
| TestWebApp | Default Server | TestWebProject |
| HelloWorldApp | Default Server | HelloWorldProject |
| optimizeit.war | Default Server | Optimizeit |
| GenerateWebServiceApp | Default Server | GenerateWebService |

Module name: *TestWebApp
Context root: *TestWebApp
Virtual Host: *default_host

***Figure 8***: *WebSphere Advanced Administrative Console displaying the deployed Web application TestWebApp*

**Note**: If you don't see **WebSphere 4.0 AE Administrator's Console** under JBuilder 8 **Tools** menu, it is because you haven't selected this option during server configuration. Click **Tools** -> **Configure Servers** -> Select **WebSphere Application Server Advanced Edition 4.0** from left-hand panel of **Configure Servers** window, click **Custom** tab from the right-hand panel, and make sure that checkbox against **Add an Administrator Console item to the Tools menu** is checked.

## Remote deployment

This section explains how to deploy an archive to a remote WebSphere Application Server.

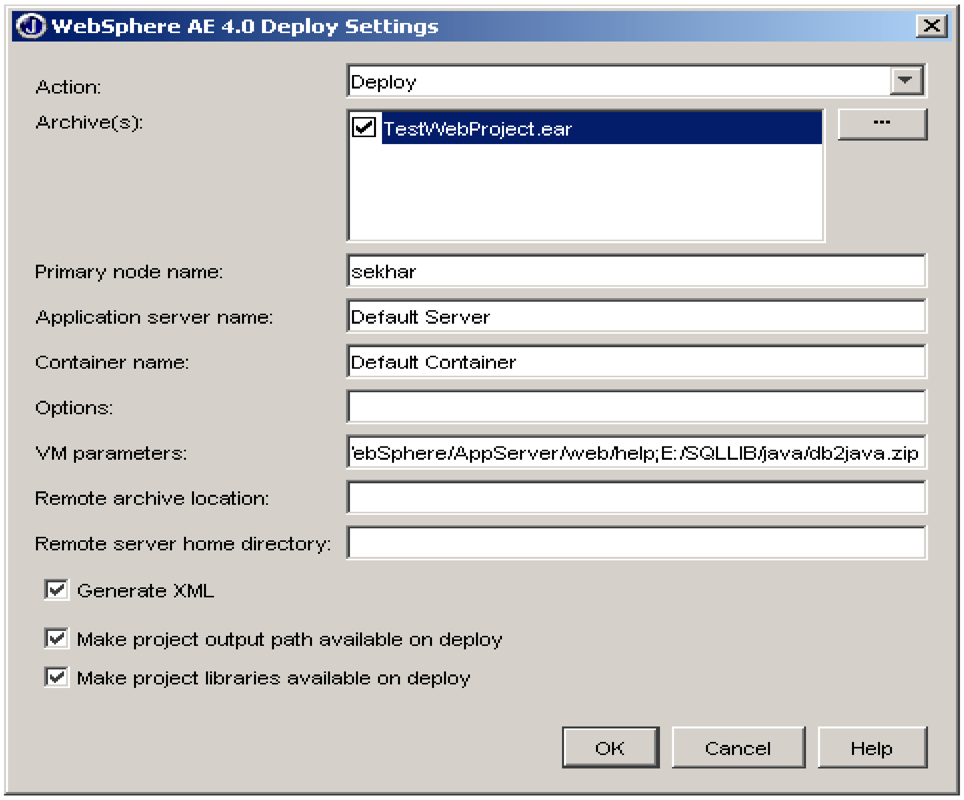
**Note**: The users need to have a local installation of the application server for the remote deployment because JBuilder need to use the deployment utilities from the local application server.

For deploying to a remote WebSphere Application Server AE 4.0, from JBuilder main menubar click **Tools** -> **Enterprise Deployment**. A window called **WebSphere AE 4.0 Deploy Settings** pops up.

Notice that the EAR **TestWebProject.ear** is already identified in the **Archive(s)** textbox.

In the **Options** field, specify -**nameServiceHost** <**host name**> -**nameServicePort** <**port number**>, where hostname and port number are the parameters of remote WebSphere Application Server. Also ensure that the **Primary node name** and the **Application server name** match the settings on the remote server.

See Figure 9 for details.



Action: Deploy
Archive(s): ☑ TestWebProject.ear
Primary node name: sekhar
Application server name: Default Server
Container name: Default Container
Options:
VM parameters: 'ebSphere/AppServer/web/help;E:/SQLLIB/java/db2java.zip
Remote archive location:
Remote server home directory:
☑ Generate XML
☑ Make project output path available on deploy
☑ Make project libraries available on deploy

***Figure 9**: JBuilder remote deployment utility tool, to deploy archives to remote WebSphere*

## Debug the JSP™

To debug the JSP, assign breakpoints in the source code wherever necessary. To assign a breakpoint, simply open the source file and single-click on that line.

Lets assign a breakpoint at line 7 of **TestJSP.jsp**. The line description is as follows.

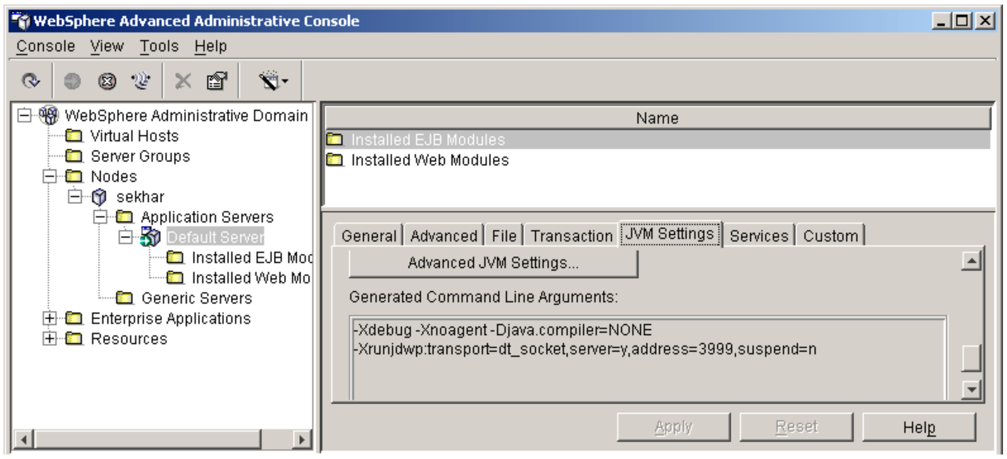
```
<jsp:useBean id="testJSPBeanId" scope="session"
     class="com.borland.demo.web.TestJSPBean" />
```

Provide the debug options to WebSphere. Start the WebSphere Server and start the **WebSphere Advanced Administrative Console**. In the console, expand **Nodes** -> expand <server_name> -> expand **Application Servers** -> click **Default Server**. Click

**JVM Settings** tab at the right-hand side, and click **Advanced JVM Settings** button. In the **command line argument** text field of **Advanced JVM Settings** window, add the following line. Then click **OK**.

```
-Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

See Figure 10 for WebSphere Console with Advanced JVM Settings.



Generated Command Line Arguments:

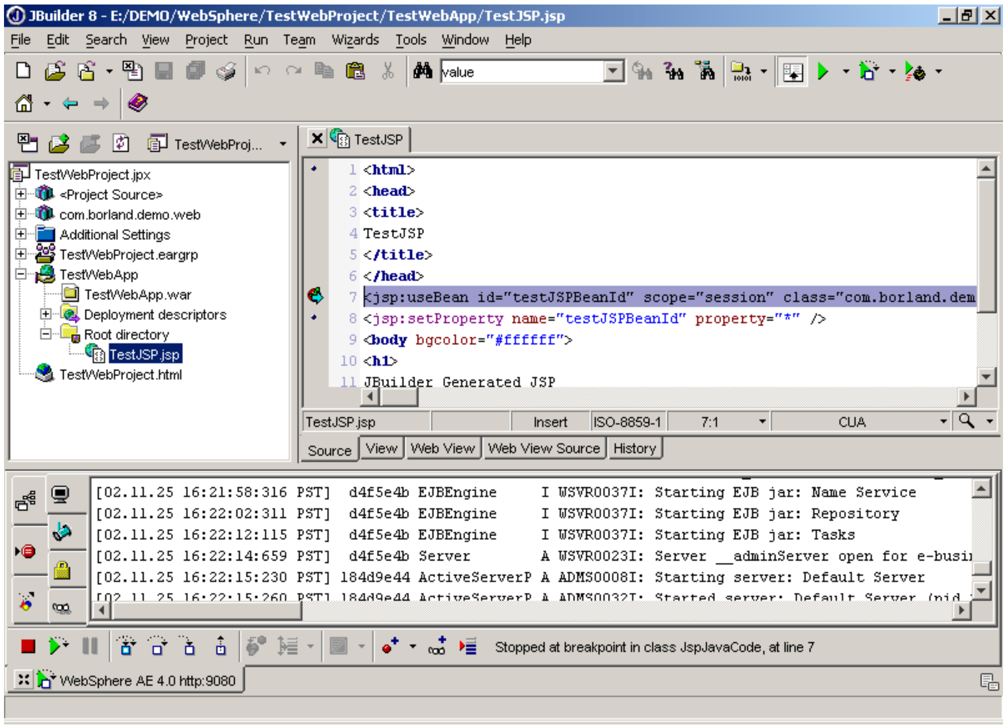
```
-Xdebug -Xnoagent -Djava.compiler=NONE
-Xrunjdwp:transport=dt_socket,server=y,address=3999,suspend=n
```

***Figure 10** : WebSphere Console: Advanced JVM setting for debugging applications in WebSphere Application Server AE 4.0*

If WebSphere is already running, stop the WebSphere Application Server. Press **Shift+F9** or just click the **Debug Project** icon. Click **TestJSP** to start WebSphere in debug mode within JBuilder. Note that JBuilder debugger has stopped at the breakpoint in JSP.

See Figure 11 for details.

**Note**: When you click **Debug Project**, JBuilder starts WebSphere Application Server in debug mode. Now users can do the debug operations from JBuilder, such as see the thread information, breakpoint information, console output, step over, step into, step out, etc.



```
<html>
<head>
<title>
TestJSP
</title>
</head>
<jsp:useBean id="testJSPBeanId" scope="session" class="com.borland.dem
<jsp:setProperty name="testJSPBeanId" property="*" />
<body bgcolor="#ffffff">
<h1>
JBuilder Generated JSP
```

```
[02.11.25 16:21:58:316 PST]  d4f5e4b EJBEngine      I WSVR0037I: Starting EJB jar: Name Service
[02.11.25 16:22:02:311 PST]  d4f5e4b EJBEngine      I WSVR0037I: Starting EJB jar: Repository
[02.11.25 16:22:12:115 PST]  d4f5e4b EJBEngine      I WSVR0037I: Starting EJB jar: Tasks
[02.11.25 16:22:14:659 PST]  d4f5e4b Server         A WSVR0023I: Server __adminServer open for e-busi
[02.11.25 16:22:15:230 PST] 184d9e44 ActiveServerP A ADMS0008I: Starting server: Default Server
[02.11.25 16:22:15:260 PST] 184d9e44 ActiveServerP A ADMS0032I: Started server: Default Server (pid
```

Stopped at breakpoint in class JspJavaCode, at line 7

***Figure 11**: WebSphere Application Server running in JBuilder in debug mode and debugger stopped at JSP breakpoint*

## Working with servlets

### Create a servlet

JBuilder provides an easy-to-use wizard to create servlets.

Lets create a servlet with the same project **TestWebProject** and same Web application **TestWebApp**.

From the JBuilder main menubar, click **File** -> **New** -> **Web** tab of **Object Gallery** -> **Servlet**. Click **OK**.

In **Servlet Wizard–Step 1 of 5**, find package name **com.borland.demo.web** from the **Package** combo box. Also enter the servlet name as **TestServlet**. Note that **TestWebApp** is automatically selected as Web application. Click **Next**.

In **Servlet Wizard–Step 2 of 5**, enter the methods that you want to use in the servlet. You can also generate an SHTML file with the servlet.

**Note**: For this exercise, make sure the checkboxes against methods **doGet()** and **doPost()** are checked. Leave other defaults and click **Next**.

Leave the defaults in **Servlet Wizard–Step 3 of 5** and **Servlet Wizard–Step 4 of 5**.

**Servlet Wizard–Step 5 of 5** allows you to create a runtime configuration for the servlet. Click the checkbox **Create a Runtime Configuration** and leave the default configuration name as **TestServlet**. At this point, since **TestJSP** runtime configuration is also created, so you can select **TestJSP** as the **Base Configuration**. Click **Finish**.

Make the project. To make the project, click **Ctrl +F9** or just click the **Make** icon from the JBuilder toolbar.

## Deploy and run the servlet

Start WebSphere Application Server within JBuilder. To start WebSphere within JBuilder, click **Run Project** icon from JBuilder toolbar and click **TestServlet**.

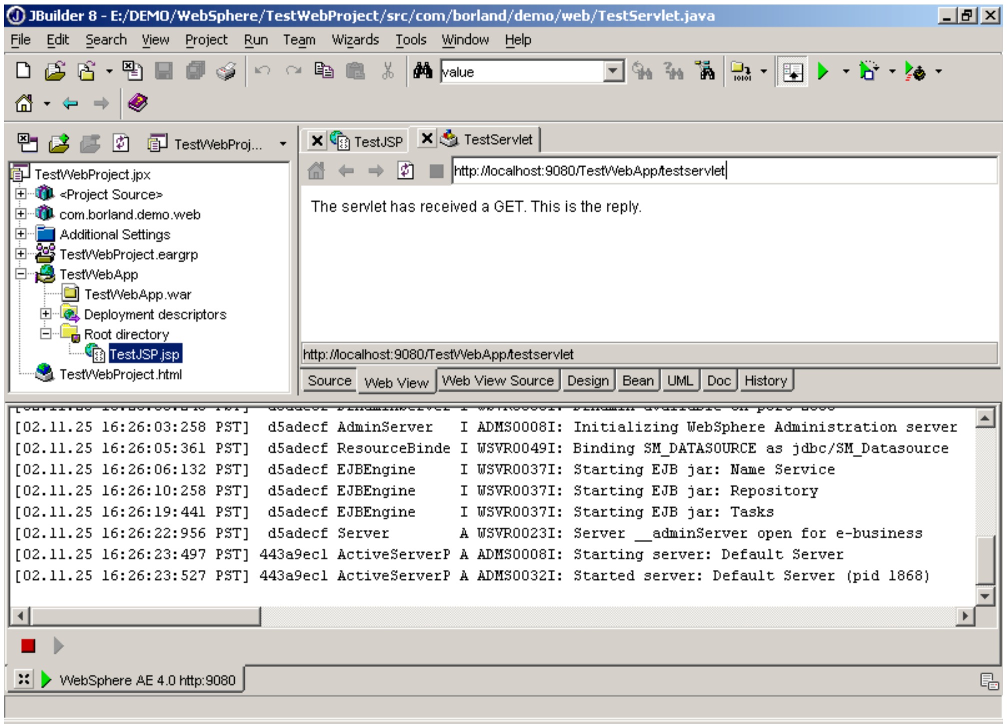
**Note**: If WebSphere is already running within JBuilder, there is no need to restart again.

Redeploy TestWebProject.ear to WebSphere. Right-click **TestWebProject.eargrp** -> click **Deploy Options for "TestWebProject.ear"** -> click **Redeploy**.

At this point, you can just right click the servlet TestServlet.java from JBuilder project tree -> click **Web Run** -> click **Use "TestServlet."** This will run the servlet and display the servlet output in the

JBuilder AppBrowser. The URL that gets executed is http://localhost:9080/TestWebApp/testservlet. You can also run this URL in your favorite browser to see the servlet output.

See Figure 12 for details.



The servlet has received a GET. This is the reply.

```
[02.11.25 16:26:03:258 PST]  d5adecf AdminServer    I ADMS0008I: Initializing WebSphere Administration server
[02.11.25 16:26:05:361 PST]  d5adecf ResourceBinde I WSVR0049I: Binding SM_DATASOURCE as jdbc/SM_Datasource
[02.11.25 16:26:06:132 PST]  d5adecf EJBEngine      I WSVR0037I: Starting EJB jar: Name Service
[02.11.25 16:26:10:258 PST]  d5adecf EJBEngine      I WSVR0037I: Starting EJB jar: Repository
[02.11.25 16:26:19:441 PST]  d5adecf EJBEngine      I WSVR0037I: Starting EJB jar: Tasks
[02.11.25 16:26:22:956 PST]  d5adecf Server         A WSVR0023I: Server __adminServer open for e-business
[02.11.25 16:26:23:497 PST] 443a9ec1 ActiveServerP A ADMS0008I: Starting server: Default Server
[02.11.25 16:26:23:527 PST] 443a9ec1 ActiveServerP A ADMS0032I: Started server: Default Server (pid 1868)
```

***Figure 12**: WebSphere running within JBuilder and displaying the server output*

## Debug the servlet

To debug the servlet **TestServlet.java**, assign breakpoints in the source code wherever necessary. To assign a breakpoint, simply open the source file and single-click the mouse on that line.

Lets assign a breakpoint at line 20 of **TestServlet.java**. The line description is as follows.

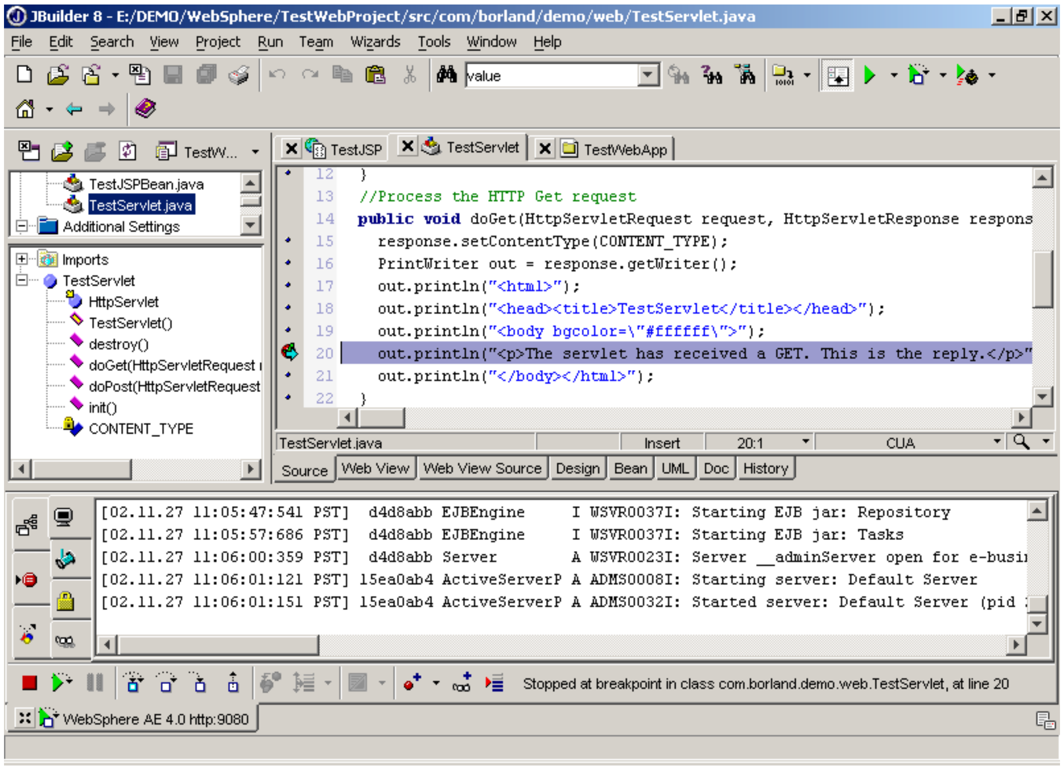
```
out.println("<p>The servlet has received a GET. This is the reply.</p>");
```

Make sure that the debug options are provided in WebSphere Console JVM settings. Refer to the section **Debug the JSP** to learn how to provide debug options to WebSphere.

If WebSphere is already running, stop the WebSphere Application Server. Press **Shift+F9** or just click the **Debug Project** icon. Click **TestServlet** to start WebSphere in debug mode within JBuilder. Note that JBuilder debugger stopped at the breakpoint in servlet.

**Note**: JBuilder put a green arrow mark at the breakpoint indicating that the debugger has stopped at that line. If there is an invalid breakpoint, JBuilder will put a cross mark against it.

See Figure 13 for details.



```
12    }
13    //Process the HTTP Get request
14    public void doGet(HttpServletRequest request, HttpServletResponse respons
15      response.setContentType(CONTENT_TYPE);
16      PrintWriter out = response.getWriter();
17      out.println("<html>");
18      out.println("<head><title>TestServlet</title></head>");
19      out.println("<body bgcolor=\"#ffffff\">");
20      out.println("<p>The servlet has received a GET. This is the reply.</p>"
21      out.println("</body></html>");
22    }
```

```
[02.11.27 11:05:47:541 PST]  d4d8abb EJBEngine      I WSVR0037I: Starting EJB jar: Repository
[02.11.27 11:05:57:686 PST]  d4d8abb EJBEngine      I WSVR0037I: Starting EJB jar: Tasks
[02.11.27 11:06:00:359 PST]  d4d8abb Server         A WSVR0023I: Server __adminServer open for e-busi
[02.11.27 11:06:01:121 PST] 15ea0ab4 ActiveServerP A ADMS0008I: Starting server: Default Server
[02.11.27 11:06:01:151 PST] 15ea0ab4 ActiveServerP A ADMS0032I: Started server: Default Server (pid
```

Stopped at breakpoint in class com.borland.demo.web.TestServlet, at line 20

***Figure 13**: WebSphere has started in debug mode within JBuilder and is debugging a servlet*

## Working with Struts

The Struts open source framework is known as the Model 2, or **Model-View Controller**, approach to software design. This framework evolved from the Model 1 design, JavaServer Pages technology. This technology offered great advances from pure servlets, where presentation HTML was coded with lengthy out.println statements in doGet() and doPut() methods. JSPs offered a way to include HTML in Java code and Java code in HTML. However, these JSPs are hard to read and hard to maintain. Both Web designers and developers are required to work in the same set of source files.

The Struts framework, first developed in 2001, combines the best of servlets and JSPs. The framework consists of a three-tiered design paradigm: the **Controller**, **Model**, and the **View**. Struts provides its own Controller component and integrates with other technologies to provide the Model and the View. Using the Struts framework is now the preferred way to create robust and long-lived professional Web applications.

For learning more about Struts, please visit http://jakarta.apache.org/struts/index.html. If you have installed JBuilder 8, you can also get information about Struts from JBuilder Help.

### Create a Struts application

Struts is the next generation Web application development. But it is difficult to write. A Struts application development cycle is probably twice or three times longer than the conventional Web application development cycle.

But JBuilder makes the Struts application development much easier for developers. The easy–to-use code wizards in JBuilder can create the Struts infrastructure efficiently and eliminate lot of hand coding. With the help of JBuilder, the developers now can focus on the business aspect of their application, rather than framework development and configuration.

While JBuilder is capable of creating very complex Struts applications, let's create a simple login application (with Struts implementation) to understand the basic guidelines.

Create a New Project in JBuilder 8 and name the project as **StrutsProject** in the directory **E:/DEMO/WebSphere/StrutsProject**.

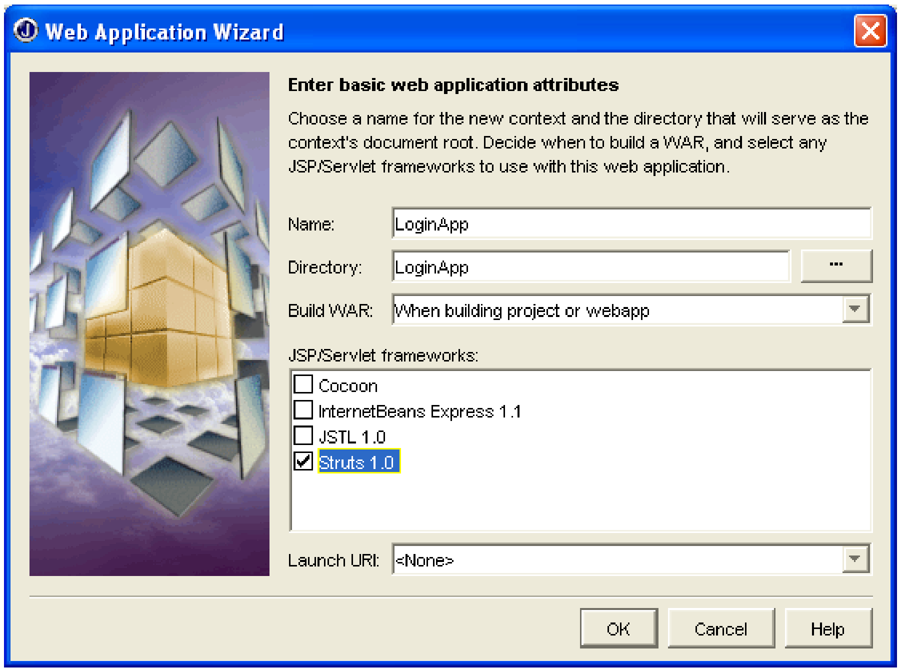
Make **WebSphere Application Server Advanced Edition 4.0** as the target server for this project. Click **Project** -> **Project Properties** -> **Server** tab and make sure that **WebSphere Application Server Advanced Edition 4.0** is selected from the combo box.

Create a Web application and enter the Web application Name and Directory as **LoginApp**. Select **Struts 1.0** under **JSP/Servlet Framework** of **Web Application Wizard**. Click **OK**.

Expand the Web application **LoginApp** in JBuilder project tree . Note that **struts-config.xml** is already created for this struts application.

Figure 14 shows the JBuilder **Web Application Wizard**, which allows the user to choose Struts 1.0.

**Note**: JBuilder 8 does not officially support the beta release of Struts 1.1. However, if you enable your struts-config.xml file for Struts 1.1 and download the Struts 1.1 .jar file to the <jbuilder>/extras folder, you will see support for Struts 1.1. The Struts Config Editor will display elements and attributes specific to 1.1. Additionally, if your Struts Web application uses a tiles.xml file for presentation, a visual editor for the Tiles configuration file is available.


Web Application Wizard

**Enter basic web application attributes**

Choose a name for the new context and the directory that will serve as the context's document root. Decide when to build a WAR, and select any JSP/Servlet frameworks to use with this web application.

Name: LoginApp

Directory: LoginApp

Build WAR: When building project or webapp

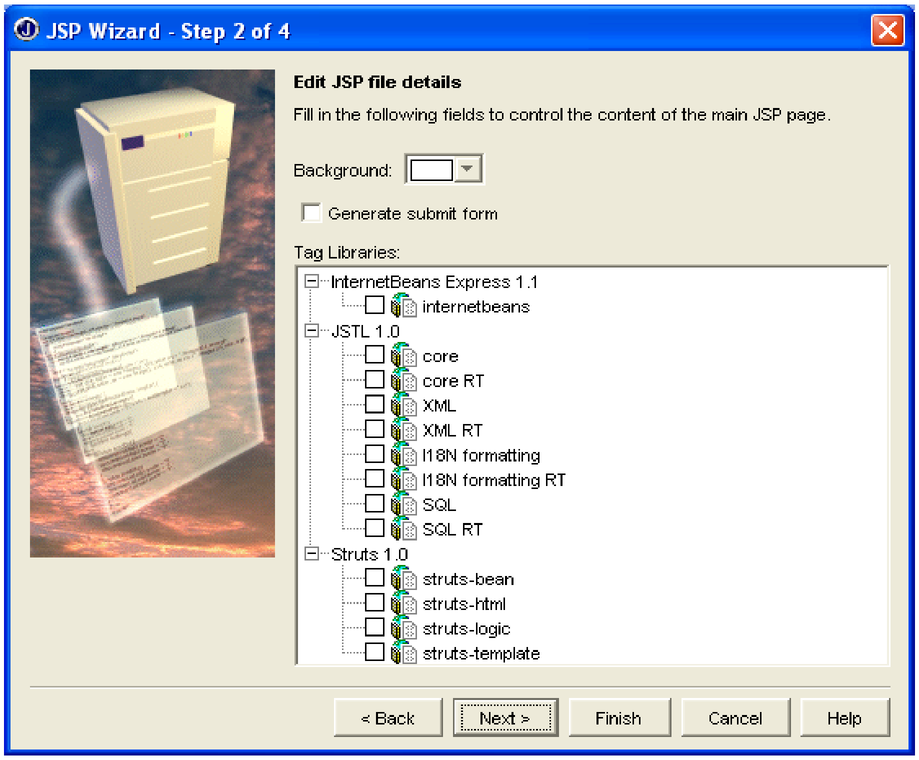
JSP/Servlet frameworks:

- [ ] Cocoon
- [ ] InternetBeans Express 1.1
- [ ] JSTL 1.0
- [✓] Struts 1.0

Launch URI: <None>

***Figure 14****: JBuilder provides wizards to create a Struts Web application*

Create two JSPs (**login.jsp** and **message.jsp**) using JBuilder **JavaServer Pages** wizard. Note that **JSP Wizard—Step 2 of 4** allows users to select struts taglibs. See Figure 15 for details.


JSP Wizard - Step 2 of 4

**Edit JSP file details**

Fill in the following fields to control the content of the main JSP page.

Background:

☐ Generate submit form

Tag Libraries:

- InternetBeans Express 1.1
  - ☐ internetbeans
- JSTL 1.0
  - ☐ core
  - ☐ core RT
  - ☐ XML
  - ☐ XML RT
  - ☐ I18N formatting
  - ☐ I18N formatting RT
  - ☐ SQL
  - ☐ SQL RT
- Struts 1.0
  - ☐ struts-bean
  - ☐ struts-html
  - ☐ struts-logic
  - ☐ struts-template

***Figure 15**: JSP wizard allows users to select the Struts taglibs*

For this exercise, don't select any tag libraries. Click **Finish** at JSP Wizard. Edit **login.jsp** and **message.jsp** in JBuilder editor and copy the following code.

**login.jsp**

```
<html>
<head>
<title>Login Page</title>
</head>
<body>
Please enter your user name and password
<br>
<form action="loginAction.do" method=POST>
<table>
<tr>
  <td>User Name:</td>
  <td><input type=text name=userName>
</tr>
<tr>
  <td>Password:</td>
  <td><input type=password name=password>
</tr>
<tr>
  <td colspan=2 align=right><input type=submit value="Login"></td>
</tr>
</table>
</form>
</body>
</html>
```

***Table 2**: login.jsp : the primary JSP for this Struts application*

**message.jsp**

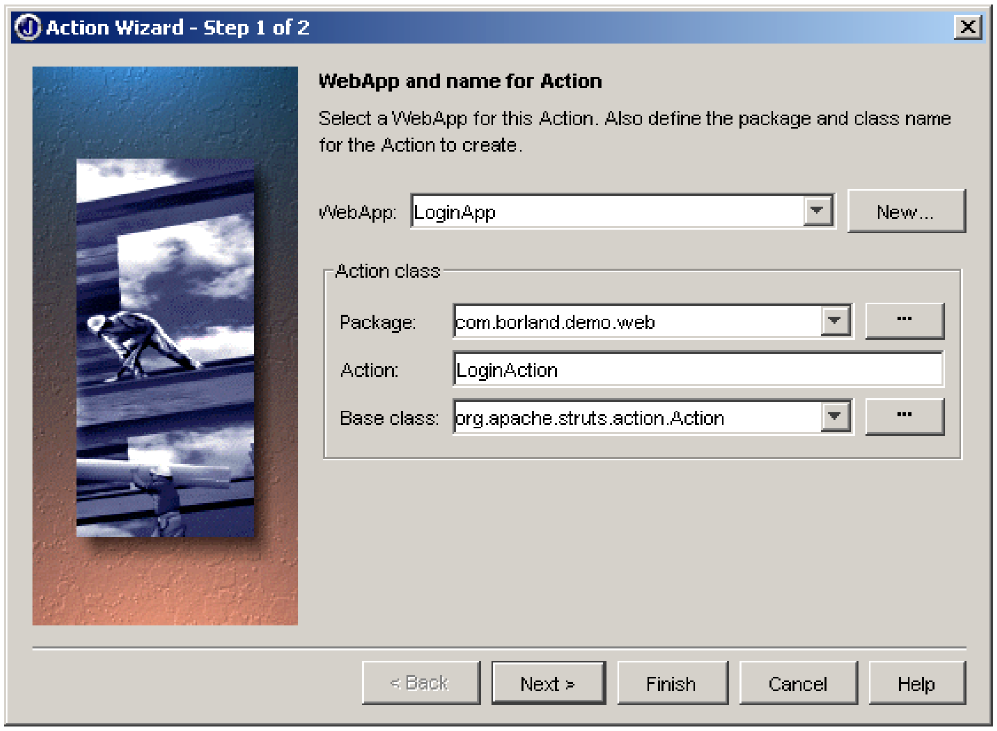
```
<html>
<head>
<title>
message
</title>
</head>
<body bgcolor="#ffffff">
<h1>
Login Successful
</h1>
</body>
</html>
```

***Table 3**: message.jsp : this JSP activates after a successful login*

**Note**: **login.jsp** is the primary JSP for this application. It accepts userName and password. **message.jsp** displays a message to the users after successful login. login.jsp and message.jsp represent the VIEW layer in this Model 2 struts application.

Create a Struts Action using the wizard from JBuilder Object Gallery. From **Object Gallery**, select **Web** tab, click **Action** and click **OK**.

In **Action Wizard–Step 1 of 2**, enter the package name as **com.borland.demo.web** and **Action** as **LoginAction**. See Figure 16 for details.



WebApp and name for Action

Select a WebApp for this Action. Also define the package and class name for the Action to create.

WebApp: LoginApp

Action class

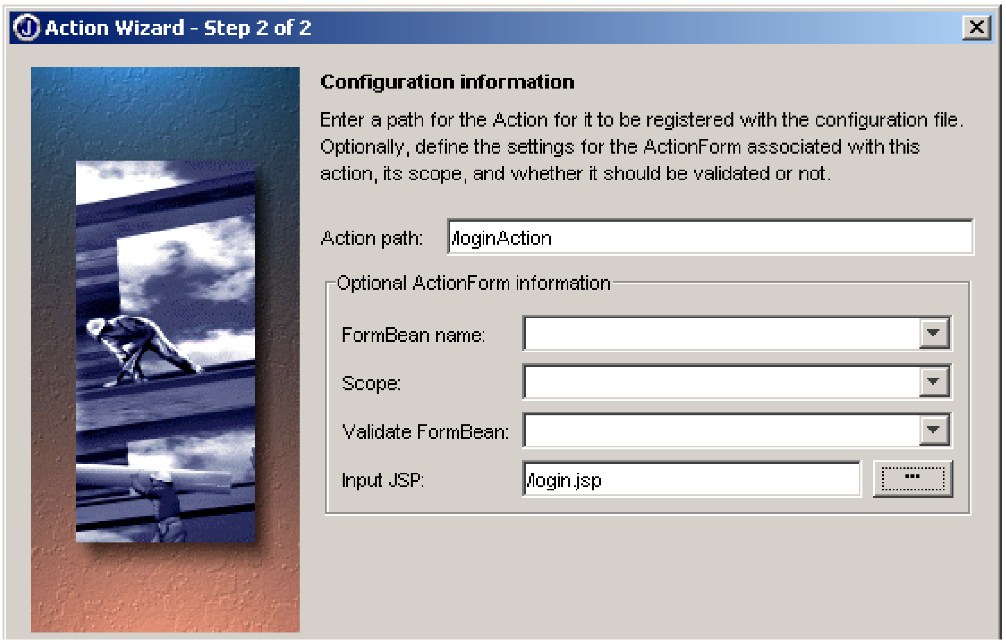
Package: com.borland.demo.web

Action: LoginAction

Base class: org.apache.struts.action.Action

***Figure 16** : JBuilder Action Wizard to create a Struts Action*

In **Action Wizard–Step 2 of 2**, click the ellipses against **Input JSP** and select login.jsp form the pop-up window. Click **Finish**. See Figure 17 for details.



**Configuration information**

Enter a path for the Action for it to be registered with the configuration file. Optionally, define the settings for the ActionForm associated with this action, its scope, and whether it should be validated or not.

Action path: /loginAction

Optional ActionForm information

FormBean name:

Scope:

Validate FormBean:

Input JSP: /login.jsp

***Figure 17***: *JBuilder Action Wizard allows the user to select the JSP for which the Struts Action has to be created*

The above wizard creates a file called **LoginAction.java**. Find the file from JBuilder project tree and edit it as shown in Table 4.

**LoginAction.java**

```
package borland.demo.struts;
import org.apache.struts.action.*;
import javax.servlet.http.*;
import javax.servlet.RequestDispatcher;
import java.io.IOException;
import javax.servlet.ServletException;
public class LoginAction extends Action {
  public ActionForward perform(ActionMapping actionMapping,
ActionForm actionForm, HttpServletRequest httpServletRequest,
    HttpServletResponse httpServletResponse) throws IOException,
ServletException {
    String userName = httpServletRequest.getParameter("userName");
    String password = httpServletRequest.getParameter("password");
    System.out.println("Username provided =" + userName);
    System.out.println("Password provided =" + password);
    if (userName!=null && password!=null &&
      userName.equals("suds") && password.equals("borland")) {
      return new ActionForward("/message.jsp");
    }
    else {
      return new ActionForward("/login.jsp");
    }
    return null;
  }
}
```

***Table 4**: LoginAction.java: implementation of action*

**Note**: The **LoginAction.java** file implements the action that is defined in the **login.jsp**. Note the following line from **login.jsp**.

```
<form action="loginAction.do" method=POST>
```

The LoginAction gets the userName and password parameters, checks the authentication, and then forwards the action to message.jsp ( upon successful login ) or login.jsp (upon unsuccessful login ). For simplicity, the authentication for userName is **suds** and password is **borland**.

The <action-mapping> is available in the **struts-config.xml** file. Take a look at this file. It demonstrates how the **loginAction.do** in the login.jsp is mapped to the LoginAction class. The **.do** is a conventional representation of the URI. Open the **web.xml** and check for the servlet-mapping for servlet-name **action**. The url-pattern for **action** is *.do. Also **action** is nothing but an instance of the Controller Servlet **org.apache.struts.action.ActionServlet**.

**struts-config.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
Struts Configuration 1.0//EN"
"http://jakarta.apache.org/struts/dtds/struts-config_1_0.dtd">
<struts-config>
  <action-mappings>
    <action type="borland.demo.struts.LoginAction" input="/login.jsp"
path="/loginAction" />
  </action-mappings>
</struts-config>
```

***Table 5**: struts-config.xml*

## Deploy and run the Struts application

Create an EAR. To create an EAR in JBuilder, click **File** -> **New** -> **Enterprise** tab of **Object Gallery** -> click **EAR** -> click **OK**. Leave the defaults in all the steps except in EAR Wizard—**Step 5 of 6**. Make sure to check the checkbox under **Include** in **EAR Wizard–Step 5 of 6**. This includes the Web module **LoginApp** to the EAR.

**Note**: You can always include additional Web and EJB modules to EAR at the later stage. Right-click the .eargrp from JBuilder project tree and click **Properties**. Click **Web** tab. JBuilder automatically identifies the available Web modules and provides an option to the users whether or not they want to include it.

Make the EAR. Start WebSphere in JBuilder and deploy the ear into WebSphere. To get help on how to start WebSphere and deploy modules into WebSphere, refer to the chapter **Working with JSPs**.

Now run the project (click **F9** or click the **Run Project** icon, then **login**). **WebSphere AE http:9080** should start within JBuilder. Also you should see the running **login.jsp** in the Borland AppBrowser.™

See Figure 18 for details.

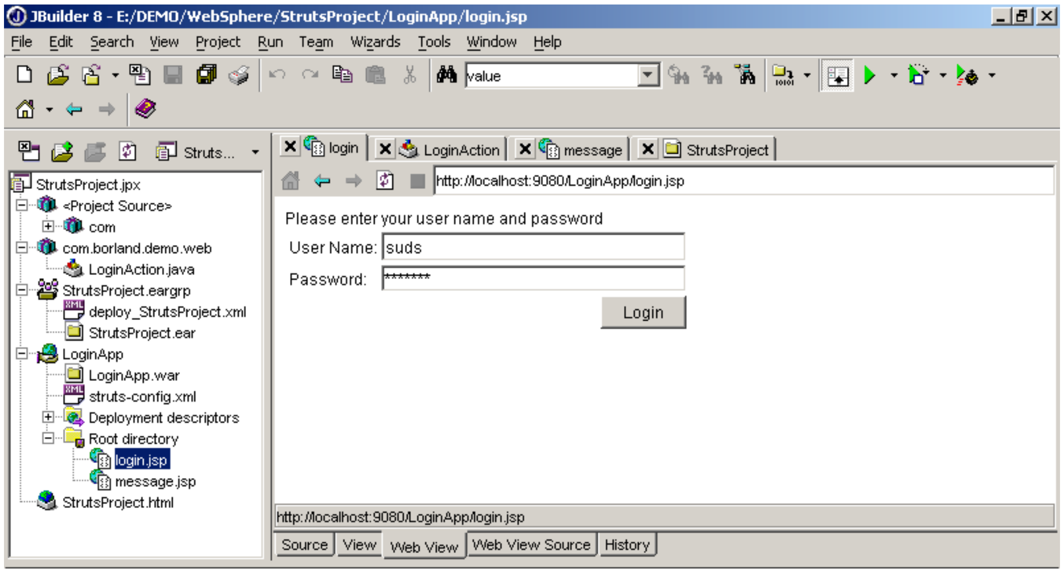


***Figure 18****: JBuilder running Struts application using WebSphere Application Server AE 4.0*

Enter the User Name as **suds** and Password as **borland**. You should see the **Login Successful** message. Try entering some other information, and it should bring the normal login.jsp screen again for re-entry of User Name and Password.

## Optimizing Web applications

JBuilder 8 Enterprises has out-of-the-box integration with Borland Optimizeit Suite.

Optimizeit Suite is a separate product available for purchase or as a trial download from the Borland Web site. Please visit the following link for downloading the latest version (version 5) of Optimizeit Suite. http://www.borland.com/products/downloads/download_optimizeit.html

Optimizeit Suite includes three different tools: Optimizeit™ Code Coverage, Optimizeit™ Profiler, and Optimizeit™ Thread Debugger. For more information, visit http://www.borland.com/optimizeit/index.html

### Code Coverage

Optimizeit Code Coverage enables developers to view the execution frequency of each class, method, and line of code, and to reduce application footprint by removing dead code.

### Profiler

Optimizeit Profiler enables developers to maximize application speed and reliability and to quickly isolate critical code that requires performance improvements.

### Thread Debugger

Optimizeit Thread Debugger helps developers to solve mysterious thread issues more easily, get the status of all threads and monitors in real time, avoid thread starvation and contentions that lead to crashes, and predict deadlocks before they occur.

**Note**: During Optimizeit Suite installation, the installer automatically detects JBuilder installation from the user's machine and prompts for integrating with JBuilder. If the user says OK, then both JBuilder and Optimizeit are configured for tight integration.

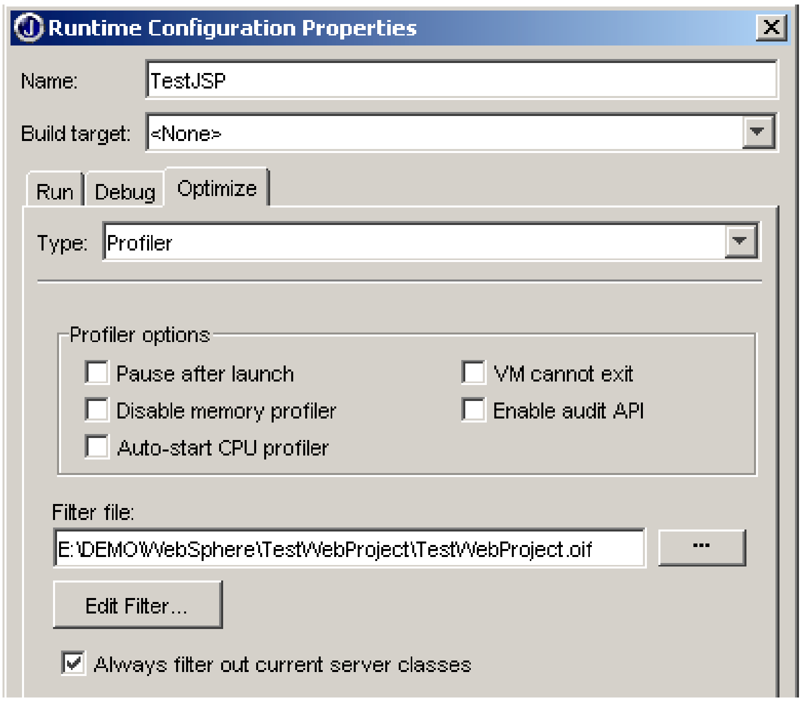
Optimizeit Suite reads the JVMs and helps the developers in profiling, thread optimizing, and code covering their applications. If the JVM runs a Java application, Optimizeit can attach to the application and display the memory, thread, and other information in a user-friendly way, which helps the developers to optimize their applications.

Optimizeit Suite is a very flexible tool, and it can attach with various market-leading IDEs and application servers. Visit the link http://info.borland.com/techpubs/optimizeit/ to find the supported J2EE™ application servers, documentation, and other information about Optimizeit Suite 5.

For this case, let's attach Optimizeit Suite with WebSphere Application Server 7.x.

Click the Optimize icon from JBuilder main menubar and click **TestJSP**. The **Runtime Configuration** window pops up and allows users to select one of the three tools (Profiler, Thread Debugger, Code Coverage). Select **Profiler** from the combo box and click **OK**.

See Figure 19 for details.



Name: TestJSP

Build target: <None>

Type: Profiler

- [ ] Pause after launch
- [ ] Disable memory profiler
- [ ] Auto-start CPU profiler
- [ ] VM cannot exit
- [ ] Enable audit API

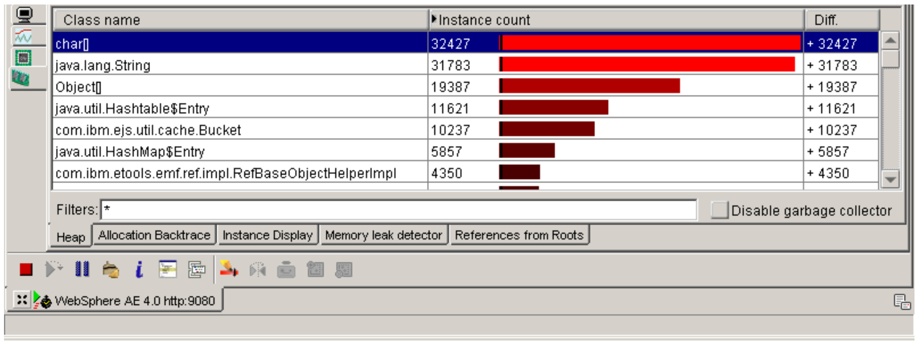
Filter file: E:\DEMO\WebSphere\TestWebProject\TestWebProject.oif

- [x] Always filter out current server classes

***Figure 19**: Runtime configuration window to choose which Optimizeit Suite tool to use*

Click **OK** and **WebSphere AE 4.0 Http 9080** starts within JBuilder 8. Note that Optimizeit Profiler is attached to WebSphere and profiling the real-time JVM information. To run your **TestJSP**, right-click the JSP from JBuilder project tree, click **WebRun**, click **TestJSP**.

See Figure 20 for details.



| Class name | Instance count | Diff. |
|---|---|---|
| char[] | 32427 | + 32427 |
| java.lang.String | 31783 | + 31783 |
| Object[] | 19387 | + 19387 |
| java.util.Hashtable$Entry | 11621 | + 11621 |
| com.ibm.ejs.util.cache.Bucket | 10237 | + 10237 |
| java.util.HashMap$Entry | 5857 | + 5857 |
| com.ibm.etools.emf.ref.impl.RefBaseObjectHelperImpl | 4350 | + 4350 |

Filters: *

- [ ] Disable garbage collector

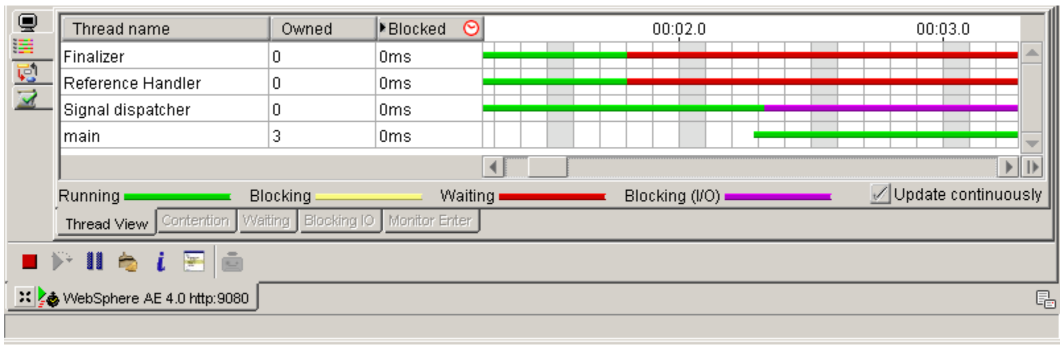
***Figure 20**: Optimizeit Profiler attached to WebSphere Application Server AE 4.0 within JBuilder 8*

To go to the normal WebSphere output window, click the **Console output, input and errors** icon on the left. Also you can click **Show Virtual Machine information** and **Show CPU profiler** for different operations.

The **Profiler** shows information about all the classes available in the JVM. If you would rather like to see the information about your application only, type **com.borland.demo.web.Test*** at the **Filters** text area. This will show you the packages started with "borland" only.

To attach Thread Debugger with WebSphere, from JBuilder main menubar, click **Run** -> **Configurations** -> select **TestJSP** -> **Edit** -> click **Optimize** -> Select **Thread Debugger** from the **Type** combo box.

See Figure 21 for Thread Debugger starting in JBuilder and attaching WebSphere.



| Thread name | Owned | Blocked |
|---|---|---|
| Finalizer | 0 | 0ms |
| Reference Handler | 0 | 0ms |
| Signal dispatcher | 0 | 0ms |
| main | 3 | 0ms |

***Figure 21**: Optimizeit Thread Debugger started in JBuilder 8 and attached with WebSphere Application Server AE 4.0*

Optimizeit Code Coverage can be started in the similar way.

There are many features available in Optimizeit Suite that help to detect critical memory leaks, performance issues, deadlocks, etc. This white paper just explained a snapshot of some of the features of Optimizeit Suite. Please read the Optimizeit Suite product documentation for information about other features.

## Troubleshooting

**Note**: The troubleshooting tips provided here are simple workarounds to understand and fix the JBuilder-WebSphere integration challenges. Borland is not responsible for supporting WebSphere related issues/fixes. Please contact WebSphere support team for other support-related questions/issues about WebSphere Application Server.

### Useful Links

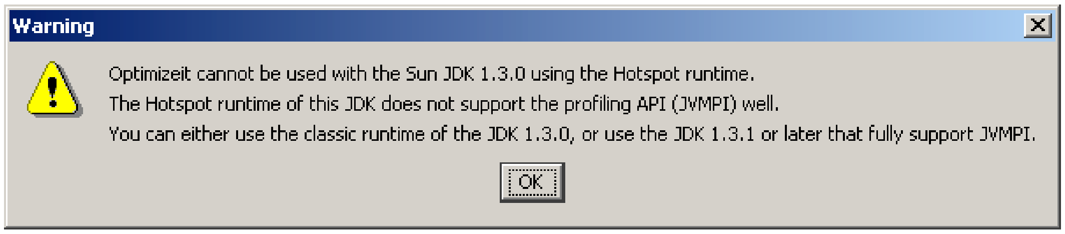
Refer to the following link for a list of bugs that is fixed in WebSphere Application Server 4.0.3 http://www-1.ibm.com/support/manager.wss?rs=180&rt=0&org=SW&doc=1052938

Refer to the following link for information and troubleshooting tips for WebSphere Application Server Advanced Edition. http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/index.html

Refer to the following link for a PDF file helping with identification of problems and providing solutions. http://www-3.ibm.com/software/webservers/appserv/doc/v40/ae/infocenter/was/pdf/nav_PDguide.pdf

### Error 1: Optimizeit™ Profiler doesn't start

#### Error description



Optimizeit cannot be used with the Sun JDK 1.3.0 using the Hotspot runtime.
The Hotspot runtime of this JDK does not support the profiling API (JVMPI) well.
You can either use the classic runtime of the JDK 1.3.0, or use the JDK 1.3.1 or later that fully support JVMPI.

***Figure 22**: Error Message while staring Optimizeit Suite tools in JBuilder*

#### When did this happen

I am trying to start Optimizeit Profiler by clicking **Optimize Project** icon from JBuilder toolbar. When I click **TestServlet** or **TestJSP** configurations, I get the above error.

### Cause

This is a known issue with JDK, and Borland is working to find a solution soon.

### Workaround

In Windows, navigate to <**user_home**>\**.jbuilder8** e.g., C:\Documents and Settings\<user_name>\.jbuilder8) and open a file called **user.properties**. Scan the file to find the lines similar as below. Replace them with the following lines. Make sure to change the installation directory of WebSphere appropriately. Also, keep a backup before making any changes.

```
optimizeit;jdkCache0.3=E:\\IBM\\WebSphere\\AppServer\\java\\bin\\javaw;3;130;1;0
optimizeit;jdkCache1.2=E:\\IBM\\WebSphere\\AppServer\\java\\bin\\javaw;3;130;1;0
optimizeit;jdkCache2.2=E:\\IBM\\WebSphere\\AppServer\\java\\bin\\javaw;3;130;1;0

A patch for this issue will be released soon in http://codecentral.borland.com
```

## Error 2: WebSphere doesn’t start

### Error description

JORB0086: Port 9000 is in use. Specify a different port number.

### When did this happen

I am trying to start WebSphere within JBuilder and getting the above error in the WebSphere AE 4.0 http:9080 tab.

### Cause

This is because WebSphere uses port 9000 for administrative purposes, and it is used by some other processes as well. If you have killed the running WebSphere already, then there may be WebSphere Java processes running in Task Manager.

### Solution

Try to find which process is using the port 9000, stop that process, and start WebSphere in JBuilder again. If you can’t stop that process, then change WebSphere administrative port number to something else.

Open Task Manager and stop the running Java processes, and restart WebSphere in JBuilder.

## Error 3: Deployment of EAR to WebSphere failed

### Error description

```
java.rmi.ServerException: RemoteException occurred in server thread; nested
           exception is:
    java.rmi.ServerException: RemoteException occurred in server thread; nested
           exception is:
    java.rmi.RemoteException: ; nested exception is:
    java.rmi.RemoteException: ; nested exception is:
    com.ibm.etools.archive.exception.DeploymentDescriptorLoadException:
           META-INF/application.xml
```

### When did this happen

I have created an EAR module successfully, started the WebSphere within JBuilder, and am trying to deploy the EAR into WebSphere. I am getting the following error in WebSphere Application Server Advanced Edition 4.0 Enterprise Deployer.

### Cause

Seems like the EAR files don't have the Web module. Probably you have missed the step to include the Web module in the EAR file.

### Solution

Check the WebSphere AE 4.0 http:9080 tab to analyze the error message. Check if your error message is something similar to the following Table 6.

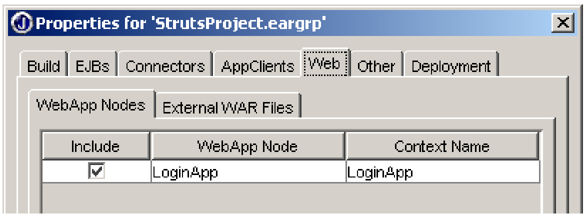
| Error Message in WebSphere AE 4.0 http:9080 |
|---|
| `1ce9945d ExceptionUtil X CNTR0020E: Non-application exception occurred while processing method getRemoteArchiveInfo on bean BeanId(admin#repository.jar#FileBrowserService, f1c72ee9f0): java.rmi.ServerException: RemoteException occurred in server thread; nested exception is:` `java.rmi.RemoteException: ; nested exception is:` |

```
   java.rmi.RemoteException: ; nested exception is:
   com.ibm.etools.archive.exception.DeploymentDescriptorLoadException: META-
         INF/application.xml
java.rmi.RemoteException: ; nested exception is:
   java.rmi.RemoteException: ; nested exception is:
   com.ibm.etools.archive.exception.DeploymentDescriptorLoadException: META-
         INF/application.xml
java.rmi.RemoteException: ; nested exception is:
   com.ibm.etools.archive.exception.DeploymentDescriptorLoadException: META-
         INF/application.xml
com.ibm.etools.archive.exception.DeploymentDescriptorLoadException: META-
         INF/application.xml
Stack trace of nested exception:
com.ibm.etools.archive.exception.ResourceLoadException: Could not load
         resource: META-INF/application.xml in archive
         E:/DEMO/WebSphere/StrutsProject/StrutsProject.ear
Stack trace of nested exception:
com.ibm.etools.archive.exception.ArchiveRuntimeException: An Exception
         occurred while parsing xml:Line #: 5:Column #: 16
Stack trace of nested exception:
org.xml.sax.SAXParseException: The content of element type "application" is
         incomplete, it must match "(icon?,display-
         name,description?,module+,security-role*)".
```

***Table 6**: Error Message in WebSphere AE 4.0 http:9080*

## Solution

From the JBuilder project tree, right-click the **eargroup .eargrp** and click **Properties**. See Figure 23 for details.



| Include | WebApp Node | Context Name |
|---|---|---|
| ✓ | LoginApp | LoginApp |

***Figure 23**: EARGRP Properties*

Click on the **Web** tab of the Properties window and make sure that the checkbox under **Include** is checked.

## Error 4: Invalidate the Web Server plugin configuration file

### Error description

1ce9945d ModuleBean A ADMR2301I: Web Server Plugin Config. The administrative action just performed may invalidate the Web Server plugin configuration file. Use the Plugin regen operation of the Node to update this file.

### When did this happen

I have started WebSphere Application Server AE 4.0 successfully in JBuilder and deployed the EAR module. I see the above error in the **WebSphere AE 4.0 http:9080** tab of the JBuilder message pane.

### Solution

This is not an error and rather a warning/information message that appears after every deployment. It is safe to regenerate the plugins when this message appears. Open WebSphere 4.0 AE Administrative Console, find the server-node, right-click server-node and click Regen WebSphere Plugin. See Figure 23 for details.

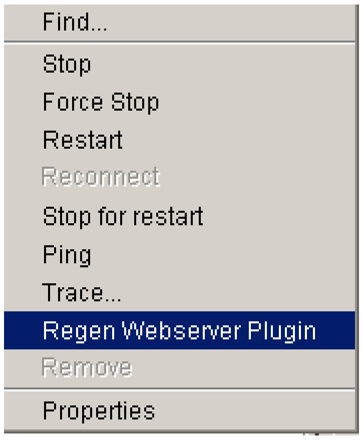


***Figure 24**: WebSphere Administrative Console: Plugin regen operation*

## Conclusion

JBuilder is a highly productive environment for Java development, deployment, testing, and optimization. Developers can increase their productivity and reduce the project lifecycle by using JBuilder as a Java application lifecycle management environment. This white paper is focused only on JSP, servlets, and Struts applications. JBuilder has many other features that make Web development a smooth experience for the developer. If you have JBuilder installed, try the tutorials available in JBuilder Help. Also refer to other white papers available in the Borland Web site for EJB and Web Services development.

## Additional information

JBuilder Online documentation
http://info.borland.com/techpubs/jbuilder/jbuilder8/index1280x1024-ent.html

OptimizeIt Online documentation
http://info.borland.com/techpubs/optimizeit/optimizeit5/index1280x1024.html

Borland JDataStore™
http://www.borland.com/jdatastore/index.html

Link to other JBuilder 8 white papers
http://www.borland.com/jbuilder/white_papers/

Borland® Enterprise Server, AppServer™ Edition
http://www.borland.com/besappserver/index.html

Sun® Microsystems Java 2 Platform Enterprise Edition
http://java.sun.com/j2ee/

JBuilder OpenTools
http://www.borland.com/jbuilder/resources/jbopentools.html

## Feedback and suggestions

Please send your feedback and suggestions to

Sudhansu Pati, Systems Engineer

spati@borland.com